

תרגיל בית: ליגת זוגי/אי-זוגי

Homework Exercise: Even/Odd League

ד"ר יורם סגל

כל הזכויות שמורות - © Dr. Segal Yoram

2025

גרסה 2.0

תוכן העניינים

12	1	מבוא: סוכני AI ופרוטוקול MCP
12	1.1	מהו סוכן AI ?
12	1.1.1	מאפיינים של סוכן AI
12	1.2	פרוטוקול Model Context Protocol – MCP
12	1.2.1	עקרונות הפרוטוקול
13	1.2.2	ארQUITטורת Host/Server
13	1.3	תחבורה HTTP על localhost
13	1.3.1	הגדרת פורטים
13	1.3.2	דוגמה לכתובות סוכן
14	1.4	מבנה הودעת JSON-RPC
14	1.5	מטרת התרגיל
14	1.5.1	יעד הלמידה
16	2	פרוטוקול הליגה הכללי
16	2.1	עקרונות הפרוטוקול
16	2.1.1	הפרדה לשולש שכבות
16	2.2	סוגי הסוכנים
16	2.2.1	מנהל ליגה – League Manager
16	2.2.2	שופט – Referee
17	2.2.3	סוכן שחקן – Player Agent
17	2.3	מזהים בפרוטוקול
17	2.3.1	דוגמאות למזהים
18	2.4	מבנה הודעה כללי – Envelope
18	2.4.1	שדות חובה במעטפת
19	2.4.2	דרישת איזור זמן – UTC/GMT
19	2.4.3	שדות אופציונליים
19	2.4.4	פורמט שדה sender
20	2.4.5	טוקן אימות – auth_token
20	2.5	זרימת הליגה הכללית
20	2.5.1	שלב 1: רישום שופטים
20	2.5.2	שלב 2: רישום שחקנים
20	2.5.3	שלב 3: יצרת לוח משחקים
20	2.5.4	שלב 4: הכרזה על מנצח

21	2.5.5 שלב 5: ניהול משחקים
21	2.5.6 שלב 6: עדכון דירוג
21	2.6 דיגרמת זרימה כללית
21	2.7 זמן תגובה – Timeouts
22	2.8 מחוזר חיים של סוכן – Agent Lifecycle
22	2.8.1 מצבי הסוכן
22	2.8.2 דיגרמת מעברי מצבים
23	2.9 טיפול בשגיאות
23	2.9.1 שגיאת ליגה – LEAGUE_ERROR
23	2.9.2 שגיאת משחק – GAME_ERROR
24	2.9.3 קוד שגיאה נפוצים
24	2.9.4 מדיניות ניסיון חזר – Retry Policy
24	2.10 תאימות גרסאות
25	2.10.1 הכרזת גרסה
25	2.10.2 מדיניות תאימות
25	2.11 עקרונות חשובים
25	2.11.1 מקור אמת ייחד
25	2.11.2 תקורת דרך Orchestrator
25	2.11.3 טיפול בכשלים
26	3 3. משחק זוגי/אי-זוגי
26	3.1 תיאור המשחק
26	3.1.1 חוקי המשחק
26	3.1.2 דוגמה למשחק
26	3.2 זרימת משחק בודד
26	3.2.1 שלב 1: הזמנה למשחק
27	3.2.2 שלב 2: אישור הגעה
27	3.2.3 שלב 3: איסוף בחרויות
27	3.2.4 שלב 4: הגרלת מספר
27	3.2.5 שלב 5: קביעת מנצח
27	3.2.6 שלב 6: דיווח תוצאה
27	3.3 מצבי המשחק
27	3.3.1 מצב WAITING_FOR_PLAYERS
28	3.3.2 מצב COLLECTING_CHOICES
28	3.3.3 מצב DRAWING_NUMBER

28	מצב FINISHED	3.3.4
28	שיטת הניקוד	3.4
28	3.4.1 ניקוד למשחק	
28	3.4.2 דירוג בליגה	
28	ליוג' Round-Robin	3.5
28	3.5.1 מספר משחקים	
29	3.5.2 לוח משחקים לדוגמה	
29	3.6 אסטרטגיות לשחקנים	3.6
29	3.6.1 אסטרטגיה אקראית	
29	3.6.2 אסטרטגיה מבוססת היסטוריה	
29	3.6.3 אסטרטגיה מונחית LLM	
30	מודול חוקי המשחק	3.7
30	3.7.1 משק המודול	
30	3.7.2 יתרון ההפרדה	
30	הרחבת למשחקים נוספים	3.8
30	3.8.1 הפשטה מהלך – GAME_MOVE	
31	3.8.2 מבנה הודעת מהלך גנרטית	
32	3.8.3 רישום סוגים משחקים – Game Registry	
32	3.8.4 יתרונות ההפשטה	
33	4 מבני הודעות JSON	
33	4.1 הודעות רישום שופט ליגה	
33	4.1.1 בקשה רישום שופט – REFEREE_REGISTER_REQUEST	
34	4.1.2 תגובה רישום שופט – REFEREE_REGISTER_RESPONSE	
35	4.2 הודעות רישום שחקן ליגה	
35	4.2.1 בקשה רישום שחקן – LEAGUE_REGISTER_REQUEST	
36	4.2.2 תגובה רישום – LEAGUE_REGISTER_RESPONSE	
37	4.3 הודעות מחזר	
37	4.3.1 הכרזת מחזר – ROUND_ANNOUNCEMENT	
37	4.4 הודעות משחק	
38	4.4.1 הזמנה למשחק – GAME_INVITATION	
38	4.4.2 אישור הגעה – GAME_JOIN_ACK	
38	4.5 הודעות בבחירה במשחק זוגי/אי-זוגי	
39	4.5.1 בקשה בבחירה – CHOOSE_PARITY_CALL	
39	4.5.2 תגובה בבחירה – CHOOSE_PARITY_RESPONSE	

40	הודעות תוצאה	4.6
40	סיום משחק – GAME_OVER –	4.6.1
41	דיווח תוצאה לliga – MATCH_RESULT_REPORT –	4.6.2
41	הודעות דירוג	4.7
42	עדכון דירוג – LEAGUE_STANDINGS_UPDATE –	4.7.1
42	הודעות סיום מחזור וliga –	4.8
43	סיום מחזור – ROUND_COMPLETED –	4.8.1
44	סיום liga – LEAGUE_COMPLETED –	4.8.2
44	הודעות שאלתא	4.9
45	שאלתא liga – LEAGUE_QUERY –	4.9.1
46	תגובה שאלתא – LEAGUE_QUERY_RESPONSE –	4.9.2
46	הודעות שגיאה	4.10
47	שגיאה ברמת הלiga – LEAGUE_ERROR –	4.10.1
48	שגיאה ברמת המשחק – GAME_ERROR –	4.10.2
48	טבלת סיכום הودעות	4.11
48	כללים חשובים	4.12
50	שדות חובה	4.12.1
50	ערכים מותרים	4.12.2
50	פורמט זמן	4.12.3

51	5 מדריך מימוש	
51	ארQUITטורה כללית	5.1
51	תרשים רכיבים	5.1.1
51	תפקיד ה-Orchestrator	5.1.2
52	מימוש שרת MCP פשוט	5.2
52	מבנה בסיסי ב-FastAPI	5.2.1
52	מימוש סוכן שחון	5.3
52	כלים נדרשים	5.3.1
53	מימוש לדוגמה	5.3.2
54	מימוש שופט	5.4
54	כלים נדרשים	5.4.1
54	רישום שופט לliga	5.4.2
55	לוגיקת קביעת מנצח	5.4.3
55	מימוש מנהל liga	5.5
55	כלים נדרשים	5.5.1

56	רישום שופט	5.5.2
57	יצירת לוח משחקים	5.5.3
58	שליחת בקשות HTTP	5.6
58	קריאה לכלי MCP	5.6.1
59	ניהול מצב	5.7
59	מצב שחזור	5.7.1
60	טיפול בשגיאות	5.8
60	זמן תגובה	5.8.1
60	תגובה לשגיאות	5.8.2
60	דפוסי חוסן (Resilience Patterns)	5.9
61	מימוש Backoff Retry עם	5.9.1
62	דפוס Circuit Breaker	5.9.2
62	תיעוד מבנה (Structured Logging)	5.10
63	מימוש Logger	5.10.1
64	דוגמה לשימוש	5.10.2
64	אימות וטוקנים (Authentication)	5.11
65	קבלת טוקן בעת רישום	5.11.1
66	שימוש בטוקן בבקשת	5.11.2
67	טיפול בשגיאות אימות	5.11.3
67	בדיקות מקומיות	5.12
67	הרצה מקומית	5.12.1
68	בדיקה חיבור	5.12.2
68	טיפים למימוש	5.13
69	דרישות תרגיל הבית	6
69	מטרת התרגיל	6.1
69	משימות חובה	6.2
69	משימה 1: מימוש סוכן שחזור	6.2.1
69	משימה 2: רישום לילגה	6.2.2
69	משימה 3: בדיקה עצמית	6.2.3
70	דרישות טכניות	6.3
70	שפת תכנות	6.3.1
70	זמן תגובה	6.3.2
70	יציבות	6.3.3
70	תהליך העבודה	6.4

70	שלב 1: פיתוח מקומי	6.4.1
70	שלב 2: ליגה פרטית	6.4.2
71	שלב 3: בדיקת תאימות עם סטודנטים אחרים	6.4.3
71	מבט לעתיד: ליגת כיתה	6.4.4
71	הגשה	6.5
71	קבצים להגשת	6.5.1
71	פורמט הגשה	6.5.2
71	דגשים כלליים לבדיקת העבודה	6.6
72	שאלות נפוצות	6.7
72	האם אפשר להשתמש בספריות חיצונית?	6.7.1
72	האם חיבים להשתמש ב-Python?	6.7.2
72	מה קורה אם הסוכן שלי קורס?	6.7.3
72	האם אפשר לעדכן את הסוכן אחורי ההגשת?	6.7.4
72	איך אדע מה הדירוג שלי?	6.7.5
73	סיכום	6.8

74	למידת MCP דרך תרגיל הליגה	7
74	השחקן כסוכן AI	7.1
74	האם סוכן השחקן הוא סוכן AI?	7.1.1
74	ארבעת המאפיינים של סוכן AI	7.1.2
74	השחקן בארכיטקטורת MCP	7.2
75	שרות או לקוח?	7.2.1
75	היחסים מול השופט ומנהל הליגה	7.2.2
75	השופט ומנהל הליגה כסוכני AI	7.3
75	סוכנים בדרגה גבוהה	7.3.1
75	שרות MCP שפועלים גם כלקחות	7.3.2
76	היפוך התפקידים: תובנה מרכזית	7.4
76	הפרדיגמה המסורתית	7.4.1
76	היפוך התפקידים בליגה	7.4.2
77	עקרון הפרדת השכבות	7.5
77	שלוש שכבות נפרדות	7.5.1
77	היתרון של הפרדה	7.5.2
78	תפקיד ה-LLM בסוכן השרת	7.6
78	הדילמה	7.6.1
78	הפתרון: הפרדת תפקידים	7.6.2

78	7.6.3	7.6. אנלוגיה: תחנת שירות ל��וחות
78	7.7	7.7. תפקיד האורקסטרטור
79	7.7.1	7.7.1. מנהל הליגה – הארכיטקט
79	7.7.2	7.7.2. השופט – המישם הדינמי
79	7.8	7.8. מה התרגיל מלמד
79	7.8.1	7.8.1. עקרונות יסוד של סוכני AI
79	7.8.2	7.8.2. עקרונות יסוד של MCP
80	7.8.3	7.8.3. חווית הלמידה
80	7.9	7.9. סיכום
81	8	8. הרכבת מערכת הליגה
81	8.1	8.1. תצורת המערכת
81	8.1.1	8.1.1. פורטים וטרמינלים
81	8.1.2	8.1.2. תפקידיו האורקסטרטוריים
82	8.2	8.2. סדר הפעלה
82	8.2.1	8.2.1. עקרון סדר הפעלה
82	8.2.2	8.2.2. טרמינל 1 – מנהל הליגה
82	8.2.3	8.2.3. טרמינלים 2-3 – שופטים
83	8.2.4	8.2.4. טרמינלים 4-7 – שחקנים
83	8.3	8.3. שלב 1: רישום שופטים
83	8.3.1	8.3.1. בקשה רישום שופט
84	8.3.2	8.3.2. תגבות מנהל הליגה
85	8.4	8.4. שלב 2: רישום שחקנים
85	8.4.1	8.4.1. בקשה רישום שחקן
86	8.4.2	8.4.2. תגבות מנהל הליגה
86	8.5	8.5. שלב 3: יצירת לוח משחקים
86	8.5.1	8.5.1. לוח משחקים לארבעה שחקנים
87	8.6	8.6. שלב 4: הכרזה על מחזור
88	8.7	8.7. שלב 5: ניהול משחק בודד
89	8.7.1	8.7.1. שלב 5.1: הזמנה למשחק
90	8.7.2	8.7.2. שלב 5.2: אישורי הגעה
92	8.7.3	8.7.3. שלב 5.3: איסוף בחירות
93	8.7.4	8.7.4. שלב 5.4: הגרלת מספר וקביעת מנצח
94	8.7.5	8.7.5. שלב 5.5: הודיעת סיום לשחקנים
95	8.7.6	8.7.6. שלב 5.6: דיווח למנהל הליגה

96	שלב 6: סיום מחזור ועדכון דירוג	8.8
99	שלב 7: סיום הליגה	8.9
99	טיפול בשגיאות	8.10
100	שגיאת אימות	8.10.1
101	שגיאת משחק – זמן תגובה	8.10.2
101	כלי שאלות זמינים	8.11
102	שאלות דירוג ממנהל הליגה	8.11.1
102	כלי נוספים	8.11.2
	דיאגרמת זרימה מלאה	103
104	טבלת תפקידי הסוכנים	8.13
104	סיכום	8.14
105	פרוטוקול נתוני הליגה	9
105	מבוא: הקוד המקורי של חברת הסוכנים	9.1
105	ארQUITטורת שלוש השכבות	9.2
105	עקרונות מנחים	9.2.1
105	שכבת הקונפיגורציה – config	9.3
106	קובץ מערכת גלובלי – config/system.json	9.3.1
106	רישום סוכנים – config/agents/agents_config.json	9.3.2
107	קונפיגורציה ליגה – config/leagues/<league_id>.json	9.3.3
107	רישום סוגי משחקים – config/games/games_registry.json	9.3.4
107	ברירות מחדל לסוכנים – config/defaults/	9.3.5
108	שכבת נתוני הריצה – data/	9.4
108	טבלת דירוג – data/leagues/<league_id>/standings.json	9.4.1
108	ההיסטוריה מחזוריים – data/leagues/<league_id>/rounds.json	9.4.2
108	נתוני משחק בודד – data/matches/<league_id>/<match_id>.json	9.4.3
109	ההיסטוריה שחקן – data/players/<player_id>/history.json	9.4.4
109	שכבת הלוגים – logs/	9.5
110	לוג ליגה מרכזי – logs/league/<league_id>/league.log.jsonl	9.5.1
110	לוג סוכן – logs/agents/<agent_id>.log.jsonl	9.5.2
110	טבלת סיכום קבצים	9.6
110	שימוש בקבצים המשותפים	9.7
112	סיכום	9.8

113	מבוא: מהקונפיגורציה לקוד	10.1
113	מבנה הספרייה	10.2
113	מודלים טיפוסיים – config_models.py –	10.3
113	הגישה: Dataclasses	10.3.1
114	מודלי קונפיגורציית מערכת	10.3.2
115	מודלי סוכנים	10.3.3
116	מודלי ליגה	10.3.4
117	טוען קונפיגורציה – ConfigLoader	10.4
117	העיקרונו: טעינה עצלה עם מטמון	10.4.1
117	שיטות הטעינה	10.4.2
118	שיטות עזר	10.4.3
118	מאגרי נתונים – Repositories	10.5
118	דפוס המאגר (Repository Pattern)	10.5.1
119	מאגר טבלת דירוג – StandingsRepository	10.5.2
119	מאגרים נוספים	10.5.3
120	רישום לוגים – JsonLogger	10.6
120	פורמט JSON Lines	10.6.1
120	מחלקת הלוגר	10.6.2
121	שיטות נוחות	10.6.3
122	שימוש בסוכנים	10.7
122	דוגמה: מנהל ליגה	10.7.1
123	דוגמה: סוכן שופט	10.7.2
124	דוגמה: רישום שגיאה	10.7.3
124	סיכום	10.8
125	מבנה הפרויקט	11
125	מבוא: מפת הדריכים	11.1
125	עץ התקיות הראשי	11.2
125	תקיות המשאבים המשותפים – /SHARED	11.3
127	תקיות הסוכנים – /agents	11.4
127	מבנה סוכן טיפוסי	11.4.1
127	תקיות התיעוד – /doc	11.5
129	דיאגרמת ארכיטקטורה	129
129	זרימת נתונים	11.6
129	קריאה וכתיבת doc	11.7
129	קריאה וכתיבה	11.7.1

130	התקנה והפעלה	11.8
130	דרישות מקדיימות	11.8.1
130	התקנת תלויות	11.8.2
130	הפעלת סוכן	11.8.3
130	רשימת קבצים מלאה	11.9
131	סיכום	11.10

133

12 **מקורות**

6 דרישות תרגיל הבית

6.1 מטרת התרגיל

בתרגיל זה תממשו סוכן שחקן לигת זוגי/אי-זוגי. בשלב זה, הסוכן שלכם ירוץ בסביבה שלכם בלבד. מומלץ לתחם עם סטודנטים אחרים כדי לוודא תאימות פרוטוקול.

חשיבות מואוד: השתמשו בפרוטוקול המוגדר במסמך זה בדיק. אחרת הסוכן שלכם לא יוכל לתקשר עם אחרים.

חובה לבנות ולתכנן את הפרויקט בכפוף להנחיות של פרק 9 (פרוטוקול נתוני הליגה), פרק 10 (ארכת כלים בפייתון), ופרק 11 (מבנה הפרויקט). כמו כן יש לוודא כי הפרויקט רץ ותפקידו כמפורט בפרק 8 (הרכבת מערכת הליגה).

תרגיל זה מבוסס על הספר:

סוכני בינה מלאכותית עם פרוטוקול הקשר המודול

AI Agents with Model Context Protocol

מאת ד"ר יורם סגל

December 9, 2025

לכן מואוד מומלץ לקרוא וללמוד את הנושא לעומק.

6.2 מישימות חובה

6.2.1 מישימה 1: מימוש סוכן שחקן

ממשו שרת MCP שמאזין על פורט ב-localhost. השרת חייב לתמוך בכלים הבאים:

- .GAME_JOIN_ACK – קבלת הזמנה למשחק והחזרת .1
- .CHOOSE_PARITY_RESPONSE – בחירת "זוגי" או "אי-זוגי" והחזרת .2
- .notify_match_result – קבלת תוצאה של משחק ועדכו מצב פנימי. .3

6.2.2 מישימה 2: רישום לигה

הסוכן חייב לשלוח בקשה רישום למנהל הליגה. הבקשה תכלול:

- שם תצוגה ייחודי (השם שלכם או כינוי).
- גרסת הסוכן.
- כתובת ה-`endpoint` של השרת.

6.2.3 מישימה 3: בדיקה עצמאית

לפני ההגשה, בדקו את הסוכן שלכם:

1. הריצו לигה מקומית עם 4 שחקנים.

2. וודאו שהסוכן מגיב לכל סוג הודעה.
3. וודאו שתבניות ה-JSON תואמות לפרטוקול.

6.3 דרישות טכניות

6.3.1 שפת תכניות

אתם יכולים לבחור כל שפה שתרצו. העיקר שהסוכן:

- ממש שרת HTTP.
- מגיב לבקשת POST בנתיב /mcp/.
- מחזיר JSON בפורמט JSON-RPC 2.0.

6.3.2 זמני תגובה

- GAME_JOIN_ACK – תוך 5 שניות.
- CHOOSE_PARITY_RESPONSE – תוך 30 שניות.
- כל תגובה אחרת – תוך 10 שניות.

6.3.3 יציבות

הסוכן חייב:

- לפעול ללא קriseות.
- לטפל בשגיאות קלט.
- לא להפסיק לפעול באמצעות ליגה.

6.4 תהליך העבודה

6.4.1 שלב 1: פיתוח מקומי

1. ממשו את הסוכן.
2. בדקו מוקומית עם הקוד שלכם.
3. תקנו באגים.

6.4.2 שלב 2: ליגה פרטיט

1. הריצו ליגה מוקומית עם 4 עותקים של הסוכן.
2. בדקו שכל התקשרותעובדת.
3. שפרו את האסטרטגייה (אופציונלי).

6.4.3 שלב 3: בדיקת תאימות עם סטודנטים אחרים

1. תאימו עם סטודנט אחר להחלפת סוכנים.
2. בדקו שהסוכנים מתקשרים ביניהם כראוי.
3. וודאו שתבניות ה-JSON תואמות לפרטוקול.

6.4.4 מבט לעתיד: ליגת כיתה

הערה חשובה

בעתיד, ייתכן שתידרשו:

- ליצור משחקים חדשים (לא רק זוגי/אי-זוגי).
- להתחרות בligaת כיתה כחלק מהפרויקט המסכם של הקורס.

נושא זה טרם נסגר וייתכנו שינויים. עלייכם להיערך לכך ולבנות את הסוכן בצורה גמישה שתאפשר הרחבה עתידית.

6.5 הגשה

6.5.1 קבצים להגשה

1. קוד מקור של הסוכן.
2. קובץ README עם הוראות הרצה.
3. דוח מפורט הכלול:
 - תיאור מלא של הארכיטקטורה והIMPLEMENTATION.
 - תיאור האסטרטגיה שנבחרה והסיבות לבחירה.
 - קשיים שנתקלתם בהם והפתרונות שמצאתם.
 - תיעוד של תהליכי הפיתוח והבדיקות.
 - מסקנות מהתרגיל והמלצות לשיפור.

6.5.2 פורמט הגשה

יש להגיש קישור לריפזיטורי כשהוא ציבורי. ויש להגיש בנהל ההגשה הרגיל כפי שהוגשו התרגילים הקודמים.

6.6 דגשים כלליים לבדיקה העבודה

מעבר לדרישות הרגילות, הクリיטריונים הבאים ייבדקו:

טבלה 13: קriterיונים לבדיקה

קriterיון	תיאור
תפקוד בסיסי	הסוכן עובד, עונה להודעות, משחק במשחקים
תאמיות פרוטוקול	tabnioniot JSON תואמות בדיק לפרטוקול
יציבות	הסוכן יציב, לא קורס, מטפל בשגיאות
aicohot kod	קוד נקי, מתועד, מאורגן
תיעוד	הוראות הרצה ברורות, תיאור מפורט
מימוש אסטרטגיה מענינית (לא רק אקראי)	אסטרטגיה

6.7 שאלות נפוצות

6.7.1 האם אפשר להשתמש בספריות חיצונית?

כן. אתם יכולים להשתמש בכל ספרייה שתרצו. וודאו שגםם מספקים הוראות התקנה.

6.7.2 האם חיבבים להשתמש ב-Python?

לא. השתמשו בכל שפה שמתאימה לכם. העיקר שהסוכן עומד בדרישות הפרוטוקול.

6.7.3 מה קורה אם הסוכן שלי קורס?

הסוכן יספג הפסד טכני במשחק הנוכחי. אם הוא לא חוזר לפעולה – הוא יוצא מהליגה.

6.7.4 האם אפשר לעדכן את הסוכן אחרי ההגשה?

לא. ההגשה סופית. בדקו היטב לפני שתם מגישים.

6.7.5 איך אדע מה הדירוג שלי?

טבלת הדירוג תפורסם לאחר כל מחזיר. תוכלם לראות את המיקום של הסוכן שלכם.

6.8 סיכום

1. מימושו סוכן שח肯 שעומד בפרוטוקול.
2. בדקנו מקומית לפני הגשה.
3. הגיעו את הקוד והדוח.
4. הסוכן שלכם ישחק בligat הכתה.

בצלחה!

מידע נוסף:

לשאלות ובירורים פנו לד"ר יורם סגל.
מומלץ לקרוא את הספר "סוכני AI עם MCP".
לפרטים נוספים על פרוטוקול MCP ראו את התיעוד הרשמי [2].

7 למידת MCP דרך תרגיל הליגה

תרגיל ליגת זוגי/אי-זוגי אינו רק תרגיל תכנות. הוא מהוות מודל פדגוגי שלם להבנת פרוטוקול MCP ועקרונות סוכני AI. בפרק זה נסביר כיצד התרגיל מלמד את עקרונות היסוד של סוכני AI ופרוטוקול MCP.

7.1 השחקן כסוכן AI

7.1.1 האם סוכן השחקן הוא סוכן AI?

השאלה הראשונה שיש לשאול היא: האם סוכן השחקן (Player Agent) בliga הוא באמת סוכן AI? התשובה היא חד-משמעות: כן.

סוכן AI מוגדר כישות המקיים אינטראקציה עם הסביבה על מנת להשיג מטרות מוגדרות [1]. להבדיל מתוכנית רגילה המבצעת הוראות קבועות מראש, סוכן AI הוא תוכנה אוטונומית שמקבלת מידע מהסביבה, מעבדת אותו, ומחילה בעצמה מה לבצע על בסיס המצב הנוכחי.

7.1.2 ארבעת המאפיינים של סוכן AI

נבחן את סוכן השחקן בliga לאור ארבעת המאפיינים העיקריים של סוכן AI:

1. **אוטונומיות** – הסוכן פועל באופן עצמאי. בהקשר המשחק, סוכן השחקן מחליט באופן אוטונומי איזו אסטרטגיה לבחור: "זוגי" (even) או "אי-זוגי" (odd). אף אחד לא אומר לו מה לבחור.

2. **תפיסה** – הסוכן קולט מידע מהסביבה. השחקן קולט הודעות הזמן למשחק ממשחקים (GAME_OVER) מהשופט ומנהל הליגה.

3. **פעולה** – הסוכן משפייע על הסביבה. השחקן מבצע פעולות על ידי שליחת בחריות (CHOOSE_PARITY_RESPONSE) ואישור הגעה (GAME_JOIN_ACK) למשחקים.

4. **תכליתיות** – יש לו מטרת מוגדרת. מטרתו היא לשחק, לנצח במשחקים ולעדכן את מצבו הפנימי, כגון היסטוריות ניצחונות והפסדים.

סוכן השחקן יכול אף להשתמש במודל שפה גדול (LLM) כדי לבחור את האסטרטגיה הטובה ביותר. בכך הוא מגדים "חשיבה" או "הסקת מסקנות" לפני ביצוע הפעולה.

7.2 השחקן בארכיטקטורת MCP

7.2.1 שרת או ל��וח?

בארQUITטורת ליגת זוגי/אי-זוגי, השחקן הוא בעיקרו **שרת MCP**. שרת MCP הוא הרכיב שהושך יכולות ושירותים, המכונים "כלי" (Tools), "כליים" (Tools), "משאבים" (Resources) או "הנחיות" (Prompts). השירות מוגדר כתהיליך נפרד הפעיל על פורט מוגדר ומספק "שער" לעולם החיצון [2].

סוכן השחקן נדרש למשר MCP שמקבל בקשות POST בנתיב `mcp/`. הכלים שהוא חושף נקראים באמצעות פרוטוקול JSON-RPC 2.0. הכלים שהשחקן מחייב למשר כוללים:

- `handle_game_invitation` – טיפול בהזמנה למשחק.
- `choose_parity` – בחירת "זוגי" או "אי-זוגי".
- `notify_match_result` – קבלת הודעה על תוצאת המשחק.

7.2.2 היחסים מול השופט ומנהל הליגה

בהינתן שהשחקן הוא שרת, מי שקורא לשירותיו הוא הלקוח (Client). במערכת הליגה, השופט (Referee) ומנהל הליגה (League Manager) הם שפועלים כלוקחות או אורקסטרטורים (Orchestrators).

השופט הוא זה שיזכר את בקשת JSON-RPC הקוראת לכל `choose_parity` של השחקן. כאשר השופט רוצה לאסוף בחירות מהשחקנים, הוא שולח בקשת `CHOOSE_PARITY_CALL` לכל שחקן.

לסייע: אף על פי שסוכן השחקן הוא סוכן AI אוטונומי, מבחינת מימוש פרוטוקול MCP, הוא ממלא את תפקיד השירות המציע יכולות לאורקסטרטורים המרכזים.

7.3 השופט ומנהל הליגה כסוכני AI

7.3.1 סוכנים בדרجة גבוהה

גם השופט ומנהל הליגה מוגדרים כסוכני AI. הם עומדים באותם ארבעה מאפיינים: הסוכנים הללו אינם פסיביים. הם מנהלים את המערכת כולה בהתאם לכללים ומטרות קבועות. זהה מוחות האוטונומיות והתכליתיות של סוכן AI.

7.3.2 שרת MCP שפועלים גם כלוקחות

שני הסוכנים הללו מוגדרים כשרת MCP:

- מנהל הליגה פועל כשרת MCP בפורט 8000. הוא מימוש כלים כמו `register_referee`, `report_match_result` ו- `register_player`.
- השופט פועל כשרת MCP בפורט 8001. הוא מימוש כלים כמו `start_match`, `collect_choices` ו-

הערה חשובה: השופט ומנהל הליגה, אף שהם מוגדרים כשרתים, חייבים לפעול גם כלוקחות MCP כדי למלא את תפקידם המרכזי. לדוגמה:

טבלה 14: מאפייני סוכן AI עבור השופט ומנהל הליגה

מאפיין	מנהל ליגה	שופט
תכליתיות	ניהול הליגה כולה, רישום שופטים וশחקנים, ייצרת לוח משחקים, חישוב דירוג	רישום למנהל הליגה, ניהול משחק בודד, אימות חוקיות מהלכים, קביעה מנצח
אוטונומיות	פועל באופן עצמאי לרישום שופטים ולקביעת סבבי משחק	נרשם באופן עצמאי לliga ומנהל את שלבי המשחק
תפיסה	קולט בקשות רישום משופטים וশחקנים, תוצאות מהשופטים	קולט אישורי הגעה, בחירות זוגיות/אי-זוגיות מהשחקנים
פעולה	מאשר רישום שופטים וশחקנים, שלוח הכרזות מחזור, מעדכן טבלאות דירוג	שלוח בקשת רישום לliga, שלוח הזמנות בבחירה, מדווח תוצאות המשחק

- השופט חייב לפעול(Clk) כדי להרשם למנהל הליגה (REGISTER_REGISTER_AI). (QUEST).
 - השופט חייב לפעול(Clk) כדי לקרוא לכלי choose_parity של סוכן השחקן.
 - מנהל הליגה חייב לפעול(Clk) כדי לשולח את הכרזות המוחזר לסטטוס השחקנים.
- במערכת זו, השרתים המרכזיים הם למעשה קוחות-אורקסטרטורים כאשר הם צריכים להניע פעולה אצל שרת השחקנים.

7.4 היפוך התפקידים: תובנה מרכזית

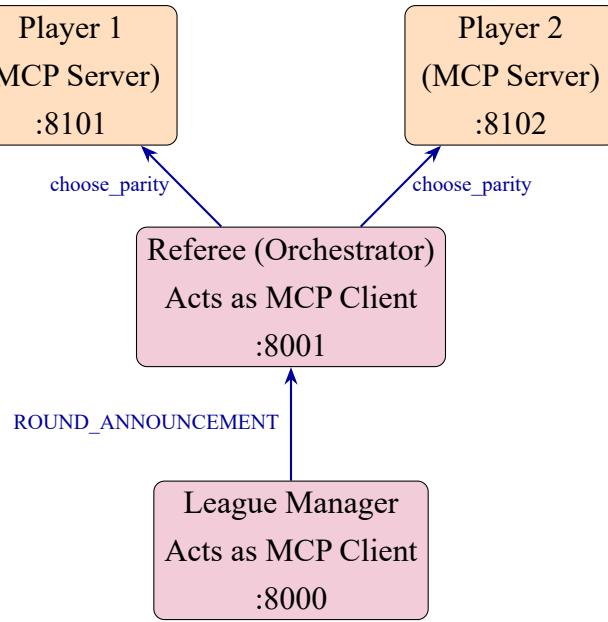
7.4.1 הפרדיגמה המסורתית

בארכיטקטורת שרת-לקוח הטיפוסית, הלקוח הוא הרכיב האקטיבי ששולח בקשות, והשרת הוא הרכיב הפסיבי שמחכה לבקשתו. בligaת ה-AI, מתקיים היפוך תפקידים ייצרתי.

7.4.2 היפוך תפקידים בליגה

השחקן (הסוכן האוטונומי) הוא השרת: למرات שהשחקן הוא הישות האוטונומית שצרכיה לבצע פעולה, הוא נדרש לחושף את יכולותיו כשרת MCP.

השופט ומנהל הליגה (אורקסטרטורים) הם הלקוחות: השופט הוא האורקסטרטור שפועל(Clk) MCP וקורא לכלי choose_parity של השחקן כדי להניע את המהלך הבא במשחק.



7.5 עקרון הפרדת השכבות

7.5.1 שלוש שכבות נפרדות

פרוטוקול MCP מאפשר הפרדה ברורה בין התפקידים:

1. **שכבת הליגה** (מנהלת על ידי מנהל הליגה) – גיוס שחקנים, לוח משחקים- (Round-robin, וטבלת דירוג).
2. **שכבת השיפוט** (מנהלת על ידי השופט) – ניהול משחק בודד וAIMOS מילכים.
3. **שכבת חוקי המשחק** (מנהלת על ידי מודול נפרד) – הלוגיקה הספציפית למשחק זוגי/אי-זוגני.

7.5.2 היתרון של ההפרדה

השחקן, בכך שהוא חושף ממשק MCP סטנדרטי (JSON-RPC 2.0 על גבי HTTP), מאפשר ליגת להישאר אגנוזטית לשפת הפיתוח או לסטרטגייה הפנימית שלו. זהו פתרון לביעית הrogrammatica שבה לכל סוכן ולכל מודול נדרשה בעבר אינטגרציה ייחודית. פרוטוקול MCP פותר זאת על ידי יצירת ממשק אוניברסלי [2].

כאשר השחקן מקבל בקשה כמו CHOOSE_PARITY_CALL, הנתונים מגיעים במבנה JSON קבוע. השחקן מגיב עמו CHOOSE_PARITY_RESPONSE, גם כן במבנה קבוע. זה מבטיח כי כל סוכן, ללא קשר לאופן שבו הוא מחשב את הנתונים, יכול לתקשר באופן עקבי עם כל אורקסטרטור אחר המכבד את הפרוטוקול.

7.6 תפקיד ה-LLM בסוכן השירות

7.6.1 הדילמה

עליה שאלת מעניינית: מצד אחד, השחקן מוגדר כשרת MCP שחושף יכולות. מצד שני, הוא מתואר כסוכן AI אוטונומי שיכל להשתמש ב-LLM כ"מוח" לבחירת אסטרטגיה. בהגדירות מסורתיות, שרת אינו מפעיל "מוח" אלא ממלא בקשה.

7.6.2 הפתרון: הפרדת תפקידיים

הפתרון טמון בהבנה שתפקיד MCP (לקוח/שרת) ומרכיבי ה-AI (מוח/כליים) הם מושגים נפרדים אך משלימים.

הסוכן הוא גם שרת וגם廉 (בפועל): כל אחד מהסוכנים הוא בפועל גם שרת וגם廉. תפקיד השירות נדרש לכל סוכן המארח את עצמו כדי לאפשר לסוכנים אחרים לקרוא לכליו. תפקיד הל��ון נדרש לכל סוכן ש策יך ליזום אינטראקציה.

ה-LLM כרכיב פנימי: מודל שפה גדול הוא "מוח" של סוכן AI. אם סוכן השחקן מaż משרת MCP, ה-LLM הוא פשוט רכיב פנימי בתוך לולאת הסוכן הכללית.

כאשר השירות מקבל בקשה choose_parity:

1. שכבת MCP (השרת) קולטת את הקשה.
2. הלוגיקה הפנימית של הסוכן (ה-LLM או אסטרטגיה אחרת) מופעלת לקביעת הבחירה.
3. שכבת MCP (השרת) שולחת את התגובה בחזרה.

ה-LLM הוא "הבינה" של השירות, והוא אינו מפר את מודל שרת-לקוח. הרעיון המרכזי ב-MCP הוא להבטיח שגם כאשר ה"מוח" נמצא בתוך השירות, התקשורות החיצונית תישאר סטנדרטית באמצעות JSON-RPC.

7.6.3 אנלוגיה: תחנת שירות ללקוחות

ניתן לדמיין את הארכיטקטורה כתחנת שירות ללקוחות:

- **MCP (פרוטוקול)** – הוא הטלפון והשפה שבה מדברים (JSON-RPC על HTTP).
 - **השחקן (שרת)** – הוא משרד השירות עם קו טלפון משלו.
 - **הסטרטגיה/LLM (מוח)** – הוא היועץ החכם היושב בתוך המשרד, שמקבל את השיחה, מחשב את המענה, ומכתיב לשכבת MCP איזו תשובה לשולח בחזרה.
- הכליים הפנימיים (ה-LLM והלוגיקה) אינם חשובים לשירות פרוטוקול MCP, אלא משרתים את הכלים הציבוריים שהסוכן חושף, כגון choose_parity.

7.7 תפקיד האורקסטרטור

7.7.1 מנהל הליגה – הארכיטקט

מנהל הליגה הוא סוכן ה-AI בדרגה הגבוהה ביותר מבחינה אסטרטגיית, המנהל את שכבות הליגה. הוא אינו מעורב בחוקי המשחק עצמו, אלא בניהול הכללי: לוח משחקים וטבלת דירוג.

יתרונות הפרדה: אם הליגה תרצה להחליף את המשחק מזוגי/אי-זוגי לאייס-עיגול-Tic-Toe, מנהל הליגה כמעט ולא ישתנה. זהה הדגמה מושלמת של עקרון הפרדת התפקידים שמקדם ה-MCP.

7.7.2 השופט – המימוש הדינמי

השופט מגלה את שכבות השיפוט. הוא אינו יודע את כללי המשחק (شمוטופלים על ידי מודול נפרד), אלא הוא אחראי על ניהול השיחה (Conversation Lifecycle) בין השחקנים. השופט מודע שהשחקנים עומדים במועד התגובה (Deadlines). הוא זה שמבצע את לולאת הסוכן החיצונית עבור השחקנים – הוא קורא לכל `choose_parity` שלהם ובכך מניע את הפעולה האוטונומית של השחקן.

MCP מאפשר את חלוקת התפקידים הדרושה: השופט ומנהל הליגה אחראים על ה"איך" (הפרוטוקול והתקשורת), בעוד השחקנים אחראים על ה"מה" (הסטרטגיה והתוכן).

7.8 מה התרגיל מלמד

7.8.1 עקרונות יסוד של סוכני AI

התרגיל מלמד את ארבעת המאפיינים של סוכן AI באופן מעשי:

- אוטונומיות – השחקן מחליט בעצמו.
- תפיסה – השחקן קולט הودעות מהמערכת.
- פעולה – השחקן שולח תשובות.
- תכליות – השחקן שואף לנצח.

7.8.2 עקרונות יסוד של MCP

התרגיל מלמד את עקרונות הליבה של פרוטוקול MCP:

1. **משחק סטנדרטי** – כל סוכן חושף כלים דרך JSON-RPC 2.0.
2. **הפרדת תפקידים** – שכבות הליגה, שכבות השיפוט, ושכבות חוקי המשחק.
3. **אגנוזטיות לשפה** – ניתן למשתמש סוכן בכל שפת תכנות.
4. **תקשורת דרך אורקסטרטור** – סוכנים לא מדברים ישירות, אלא דרך השופט או מנהל הליגה.
5. **רישום סוכנים** – גם שופטים וגם שחקנים נרשמים למנהל הליגה לפני תחילת המשחקים.

7.8.3 חווית הלמידה

בסיום התרגיל, הסטודנט יבין:

- כיצד סוכן AI מתקשר עם סוכנים אחרים.
- כיצד לבנות שרת MCP פשוט.
- מהי משמעות "הכליים" (Tools) בפרוטוקול MCP.
- כיצד אורקסטרטור מנהל אינטראקציה בין סוכנים.
- מדוע הפרדת שכבות חשובה לתכנון מערכות AI.

7.9 סיכום

תרגיל ליגת זוגי/אי-זוגי מהוות מודל פדגוגי מושלם להבנת פרוטוקול MCP וסוכני AI. המשחק פשוט מאפשר להתמקד בעקרונות הארכיטקטוניים מביי להסתבך בלוגיקה מורכבת.

הסטודנט לומד שסוכן AI יכול להיות גם שרת MCP – היפוך תפקדים יצירתי המאפשר לאורקסטרטור לקרוא לסטודנטים ולהניע את פעולתם. הפרדה לשכבות מבטיחה שנייתן להחליף את משחק הליגה בעתיד מביי לשנות את הפרוטוקול הכללי.

לפרטים נוספים על פרוטוקול MCP, ראו את הספר "סוכני AI עם MCP" [1] ואת התיעוד הרשמי של Anthropic [2].

8 הרצת מערכת הליגה

נספח זה מציג הדרכה מעשית להרצת מערכת הליגה המלאה. נדגים כיצד להפעיל את כל הסוכנים ולנהל לигה עם מנהל ליגה אחד, שני שופטים וארבעה שחקנים. הדוגמאות מבוססות על פרוטוקול league.v2 המתוואר בפרקים הקודמים.

8.1 תצורת המערכת

8.1.1 פורטים וטרמינלים

כל סוכן במערכת פועל כשרת HTTP נפרד על פורט ייחודי ב-localhost. בדוגמה זו נריצ' 7 טרמינלים:

טבלה 15: הקצת פורטים וטרמינלים

	טרמינל	סוכן	פורט	Endpoint
1		מנהל ליגה	8000	http://localhost:8000/mcp
2		שופט REF01	8001	http://localhost:8001/mcp
3		שופט REF02	8002	http://localhost:8002/mcp
4		שחקן P01	8101	http://localhost:8101/mcp
5		שחקן P02	8102	http://localhost:8102/mcp
6		שחקן P03	8103	http://localhost:8103/mcp
7		שחקן P04	8104	http://localhost:8104/mcp

8.1.2 תפקידו של אורתקסטרטור

במערכת שני סוגי Orchestrator

- **מנהל הליגה** – Orchestrator עליון של הליגה. הוא מקור האמת לטבלת הדירוג, לוח המשחקים ומצב המחזוריים.

- **שופטים** – Orchestratorים מקומיים למשחק בודד. כל שופט הוא מקור האמת למצב המשחק שלו.

8.2 סדר הפעלה

8.2.1 עקרון סדר הפעלה

סדר הפעלה קריטי לתפקוד תקין של המערכת:

1. **מנהל הליגה** – חייב לעלות ראשון.
2. **שופטים** – בעליים ונרשמים למנהל הליגה.
3. **שחקנים** – בעליים ונרשמים למנהל הליגה.
4. **תחלת הליגה** – רק לאחר השלמת כל הרישומים.

8.2.2 טרמינל 1 – מנהל הליגה

הפעלה מנהל הליגה

```
# Terminal 1 - League Manager
python league_manager.py # Listening on :8000
```

מנהל הליגה מזין לבקשת POST בכתובת <http://localhost:8000/mcp>

8.2.3 טרמינלים 2-3 – שופטים

הפעלה שופטים

```
# Terminal 2 - Referee Alpha
python referee.py --port 8001

# Terminal 3 - Referee Beta
python referee.py --port 8002
```

כל שופט, בעת עלייתו, מפעיל פונקציה `register_to_league` ששולחת REF-ERE_EEE_REGISTER_REQUEST למנהל הליגה.

הפעלת שחקנים

```
# Terminal 4 - Player P01
python player.py --port 8101

# Terminal 5 - Player P02
python player.py --port 8102

# Terminal 6 - Player P03
python player.py --port 8103

# Terminal 7 - Player P04
python player.py --port 8104
```

כל שחקן שולח LEAGUE_REGISTER_REQUEST למנהל הליגה.

8.3 שלב 1: רישום שופטים

כל שופט, מיד עם עליית השרת שלו, קורא מצד הלוקוח למנהל הליגה.

8.3.1 בקשה רישום שופט**REFEREE_REGISTER_REQUEST – בקשה רישום שופט**

```
{
  "jsonrpc": "2.0",
  "method": "register_referee",
  "params": {
    "protocol": "league.v2",
    "message_type": "REFEREE_REGISTER_REQUEST",
    "sender": "referee:alpha",
    "timestamp": "2025-01-15T10:00:00Z",
    "conversation_id": "conv-ref-alpha-reg-001",
    "referee_meta": {
      "display_name": "Referee\u0327Alpha",
      "version": "1.0.0",
      "game_types": ["even_odd"],
      "contact_endpoint": "http://localhost:8001/mcp",
      "max_concurrent_matches": 2
    }
  },
  "id": 1
}
```

REFEREE_REGISTER_RESPONSE – תגובה רישום שופט

```
{  
  "jsonrpc": "2.0",  
  "result": {  
    "protocol": "league.v2",  
    "message_type": "REFEREE_REGISTER_RESPONSE",  
    "sender": "league_manager",  
    "timestamp": "2025-01-15T10:00:01Z",  
    "conversation_id": "conv-ref-alpha-reg-001",  
    "status": "ACCEPTED",  
    "referee_id": "REF01",  
    "auth_token": "tok-ref01-abc123",  
    "league_id": "league_2025_even_odd",  
    "reason": null  
  },  
  "id": 1  
}
```

השופט השני (על פורט 8002) שולח בקשה דומה ומקבל "REF02"

8.4 שלב 2: רישום שחקנים

לאחר שהשופטים נרשמו, כל שחקן שולח בקשה רישום.

8.4.1 בקשה רישום שחקן

LEAGUE_REGISTER_REQUEST - בקשה רישום שחקן

```
{  
  "jsonrpc": "2.0",  
  "method": "register_player",  
  "params": {  
    "protocol": "league.v2",  
    "message_type": "LEAGUE_REGISTER_REQUEST",  
    "sender": "player:alpha",  
    "timestamp": "2025-01-15T10:05:00Z",  
    "conversation_id": "conv-player-alpha-reg-001",  
    "player_meta": {  
      "display_name": "AgentAlpha",  
      "version": "1.0.0",  
      "game_types": ["even_odd"],  
      "contact_endpoint": "http://localhost:8101/mcp"  
    }  
  },  
  "id": 1  
}
```

league_register_response - תגובה רישום שחקן

```
{
  "jsonrpc": "2.0",
  "result": {
    "protocol": "league.v2",
    "message_type": "LEAGUE_REGISTER_RESPONSE",
    "sender": "league_manager",
    "timestamp": "2025-01-15T10:05:01Z",
    "conversation_id": "conv-player-alpha-reg-001",
    "status": "ACCEPTED",
    "player_id": "P01",
    "auth_token": "tok-p01-xyz789",
    "league_id": "league_2025_even_odd",
    "reason": null
  },
  "id": 1
}
```

בأوبن דומה:

- שחקן על פורט 8102 מקבל "P02"
- שחקן על פורט 8103 מקבל "P03"
- שחקן על פורט 8104 מקבל "P04"

מנהל הליגה שומר מפה `referee_id` ומספר `contact_endpoint` → `player_id`
`player_id` → `contact_endpoint`

8.5 שלב 3: ייצרת לוח משחקים

לאחר שכל השחקנים והשופטים נרשמו, מנהל הליגה מפעיל לוגיקת `create_schedule` מנהל הליגה מפעיל לוגיקת `Round-Robin` על רשימת השחקנים ל-`Round-Robin`.

8.5.1 לוח משחקים לארבעה שחקנים

טבלה 16: לוח משחקים Round-Robin לארבעה שחקנים

Match ID	Player A	Player B
R1M1	P01	P02
R1M2	P03	P04
R2M1	P03	P01
R2M2	P04	P02
R3M1	P04	P01
R3M2	P03	P02

8.6 **שלב 4: הכרזה על מחזור**

מנהל הליגה שולח לכל השחקנים הודעה .ROUND_ANNOUNCEMENT

הכרזה על מחזור – ROUND_ANNOUNCEMENT

```
{  
  "jsonrpc": "2.0",  
  "method": "notify_round",  
  "params": {  
    "protocol": "league.v2",  
    "message_type": "ROUND_ANNOUNCEMENT",  
    "sender": "league_manager",  
    "timestamp": "2025-01-15T10:10:00Z",  
    "conversation_id": "conv-round-1-announce",  
    "league_id": "league_2025_even_odd",  
    "round_id": 1,  
    "matches": [  
      {  
        "match_id": "R1M1",  
        "game_type": "even_odd",  
        "player_A_id": "P01",  
        "player_B_id": "P02",  
        "referee_endpoint": "http://localhost:8001/mcp"  
      },  
      {  
        "match_id": "R1M2",  
        "game_type": "even_odd",  
        "player_A_id": "P03",  
        "player_B_id": "P04",  
        "referee_endpoint": "http://localhost:8001/mcp"  
      }  
    ],  
    "id": 10  
  }  
}
```

מרגע שנשלחה ROUND_ANNOUNCEMENT – המחזור התחליל לוגית. המשחקים עצם מתחילה רק כאשר השופט מזמין את המשתתפים.

8.7 שלב 5: ניהול משחק בודד

נתרן את זרימת המשחק R1M1: שחקן P01 נגד שחקן P02, שופט REF01.

השופט מעביר את מצב המשחק ל-WAITING_FOR_PLAYERS ושולח GAME_INVITATION לכל שחקן.

הזמנה ל-P01:

P01-ל GAME_INVITATION – הזמנה למשחק

```
{  
  "jsonrpc": "2.0",  
  "method": "handle_game_invitation",  
  "params": {  
    "protocol": "league.v2",  
    "message_type": "GAME_INVITATION",  
    "sender": "referee:REF01",  
    "timestamp": "2025-01-15T10:15:00Z",  
    "conversation_id": "conv-r1m1-001",  
    "auth_token": "tok-ref01-abc123",  
    "league_id": "league_2025_even_odd",  
    "round_id": 1,  
    "match_id": "R1M1",  
    "game_type": "even_odd",  
    "role_in_match": "PLAYER_A",  
    "opponent_id": "P02"  
  },  
  "id": 1001  
}
```

הזמנה ל-P02:

P02-ל GAME_INVITATION - הזמנה למשחק

```
{  
  "jsonrpc": "2.0",  
  "method": "handle_game_invitation",  
  "params": {  
    "protocol": "league.v2",  
    "message_type": "GAME_INVITATION",  
    "sender": "referee:REF01",  
    "timestamp": "2025-01-15T10:15:00Z",  
    "conversation_id": "conv-r1m1-001",  
    "auth_token": "tok-ref01-abc123",  
    "league_id": "league_2025_even_odd",  
    "round_id": 1,  
    "match_id": "R1M1",  
    "game_type": "even_odd",  
    "role_in_match": "PLAYER_B",  
    "opponent_id": "P01"  
  },  
  "id": 1002  
}
```

8.7.2 שלב 2: אישורי הגעה

כל שחקן מוחזיר GAME_JOIN_ACK תוך 5 שניות.

אישור מ-P01:

אישור הגעה - P01 GAME_JOIN_ACK

```
{  
  "jsonrpc": "2.0",  
  "result": {  
    "protocol": "league.v2",  
    "message_type": "GAME_JOIN_ACK",  
    "sender": "player:P01",  
    "timestamp": "2025-01-15T10:15:01Z",  
    "conversation_id": "conv-r1m1-001",  
    "auth_token": "tok-p01-xyz789",  
    "match_id": "R1M1",  
    "player_id": "P01",  
    "arrival_timestamp": "2025-01-15T10:15:01Z",  
    "accept": true  
  },  
  "id": 1001  
}
```

אישור מ-P02 GAME_JOIN_ACK

אישור הגעה - P02 GAME_JOIN_ACK

```
{  
  "jsonrpc": "2.0",  
  "result": {  
    "protocol": "league.v2",  
    "message_type": "GAME_JOIN_ACK",  
    "sender": "player:P02",  
    "timestamp": "2025-01-15T10:15:02Z",  
    "conversation_id": "conv-r1m1-001",  
    "auth_token": "tok-p02-def456",  
    "match_id": "R1M1",  
    "player_id": "P02",  
    "arrival_timestamp": "2025-01-15T10:15:02Z",  
    "accept": true  
  },  
  "id": 1002  
}
```

כasher hashofet kibbul shni ACK chiyobim bezman hmotter, hoa meuber at matzav hameshuk .COLLECTING_CHOICES-L

השופט שולח שולח CHOOSE_PARITY_CALL לכל שחקן.

בקשת בחירה ל-P01:

P01-ל CHOOSE_PARITY_CALL - בקשת בחירה

```
{
  "jsonrpc": "2.0",
  "method": "choose_parity",
  "params": {
    "protocol": "league.v2",
    "message_type": "CHOOSE_PARITY_CALL",
    "sender": "referee:REF01",
    "timestamp": "2025-01-15T10:15:05Z",
    "conversation_id": "conv-r1m1-001",
    "auth_token": "tok-ref01-abc123",
    "match_id": "R1M1",
    "player_id": "P01",
    "game_type": "even_odd",
    "context": {
      "opponent_id": "P02",
      "round_id": 1,
      "your_standings": {
        "wins": 0,
        "losses": 0,
        "draws": 0
      }
    },
    "deadline": "2025-01-15T10:15:35Z"
  },
  "id": 1101
}
```

תגובה P01 (בחר "even"):

תגובה בחירה - מ-P01 CHOOSE_PARITY_RESPONSE

```
{  
  "jsonrpc": "2.0",  
  "result": {  
    "protocol": "league.v2",  
    "message_type": "CHOOSE_PARITY_RESPONSE",  
    "sender": "player:P01",  
    "timestamp": "2025-01-15T10:15:10Z",  
    "conversation_id": "conv-r1m1-001",  
    "auth_token": "tok-p01-xyz789",  
    "match_id": "R1M1",  
    "player_id": "P01",  
    "parity_choice": "even"  
  },  
  "id": 1101  
}
```

תגובה P02 (בחירה "odd"):

תגובה בחירה - מ-P02 CHOOSE_PARITY_RESPONSE

```
{  
  "jsonrpc": "2.0",  
  "result": {  
    "protocol": "league.v2",  
    "message_type": "CHOOSE_PARITY_RESPONSE",  
    "sender": "player:P02",  
    "timestamp": "2025-01-15T10:15:12Z",  
    "conversation_id": "conv-r1m1-001",  
    "auth_token": "tok-p02-def456",  
    "match_id": "R1M1",  
    "player_id": "P02",  
    "parity_choice": "odd"  
  },  
  "id": 1102  
}
```

כאשר שתי הבחירה התקבלו נכון ובזמן, השופט מעביר את מצב המשחק ל-DRAWING_NUMBER.

8.7.4 שלב 5.4: הגרלת מספר וקבעת מנץ'

השופט מגריל מספר בין 1 ל-10, למשל 8. הוא מפעיל את מודול חוקי המשחק:

```
drawn_number = 8 -
```

number_parity = "even" -
 - בחרית 1 → "even" = P01 -
 - בחרית 2 → "odd" = P02 -
 winner_player_id = "P01" -
 status = "WIN" -
 מצב המשחק עובר ל-FINISHED.

8.7.5 שלב 5.5: הודעת סיום לשחקנים

השופט שולח GAME_OVER לשני השחקנים:

סיום משחק - GAME_OVER

```

{
  "jsonrpc": "2.0",
  "method": "notify_match_result",
  "params": {
    "protocol": "league.v2",
    "message_type": "GAME_OVER",
    "sender": "referee:REF01",
    "timestamp": "2025-01-15T10:15:30Z",
    "conversation_id": "conv-r1m1-001",
    "auth_token": "tok-ref01-abc123",
    "match_id": "R1M1",
    "game_type": "even_odd",
    "game_result": {
      "status": "WIN",
      "winner_player_id": "P01",
      "drawn_number": 8,
      "number_parity": "even",
      "choices": {
        "P01": "even",
        "P02": "odd"
      },
      "reason": "P01 chose even, number was 8 (even)"
    }
  },
  "id": 1201
}
  
```

כל שחקן מעדכן מצב פנימי (סטטיטיות, היסטוריה) ומחזיר תשובה כללית.

השופט מדוח דיווח למנהל הליגה: MATCH_RESULT_REPORT

DIVISION RESULT REPORT - דיווח תוצאה

```
{
  "jsonrpc": "2.0",
  "method": "report_match_result",
  "params": {
    "protocol": "league.v2",
    "message_type": "MATCH_RESULT_REPORT",
    "sender": "referee:REF01",
    "timestamp": "2025-01-15T10:15:35Z",
    "conversation_id": "conv-r1m1-report",
    "auth_token": "tok-ref01-abc123",
    "league_id": "league_2025_even_odd",
    "round_id": 1,
    "match_id": "R1M1",
    "game_type": "even_odd",
    "result": {
      "winner": "P01",
      "score": {
        "P01": 3,
        "P02": 0
      },
      "details": {
        "drawn_number": 8,
        "choices": {
          "P01": "even",
          "P02": "odd"
        }
      }
    },
    "id": 1301
  }
}
```

מנהל הליגה מעדרן טבלת נקודות בהתאם לטבלת הניקוד (ニitchon = 3 נקודות).

8.8 שלב 6: סיום מחזור ועקבון דירוג

מחזור מס' 1 מסתיים כאשר לכל מושקיע המחזור התקבל MATCH_RESULT_REPORT מנהל הליגה:

1. מכריז שהמחזור סגור (אפשר לעבור ל-round_id=2).
2. מחשב טבלת דירוג: played ,losses ,draws ,wins ,points לכל שחקן.
3. שולח לכל השחקנים הודעה LEAGUE_STANDINGS_UPDATE.

עדכון דירוג – LEAGUE_STANDINGS_UPDATE

```
{  
  "jsonrpc": "2.0",  
  "method": "update_standings",  
  "params": {  
    "protocol": "league.v2",  
    "message_type": "LEAGUE_STANDINGS_UPDATE",  
    "sender": "league_manager",  
    "timestamp": "2025-01-15T10:20:00Z",  
    "conversation_id": "conv-round-1-standings",  
    "league_id": "league_2025_even_odd",  
    "round_id": 1,  
    "standings": [  
      {  
        "rank": 1,  
        "player_id": "P01",  
        "display_name": "Agent\u2022Alpha",  
        "played": 1,  
        "wins": 1,  
        "draws": 0,  
        "losses": 0,  
        "points": 3  
      },  
      {  
        "rank": 2,  
        "player_id": "P03",  
        "display_name": "Agent\u2022Gamma",  
        "played": 1,  
        "wins": 0,  
        "draws": 1,  
        "losses": 0,  
        "points": 1  
      },  
      {  
        "rank": 3,  
        "player_id": "P04",  
        "display_name": "Agent\u2022Delta",  
        "played": 1,  
        "wins": 0,  
        "draws": 1,  
        "losses": 0,  
        "points": 1  
      },  
      {  
        "rank": 4,  
        "player_id": "P02",  
        "display_name": "Agent\u2022Beta",  
        "played": 1,  
        "wins": 0,  
        "draws": 0,  
        "losses": 1,  
        "points": 0  
      }  
    ]  
  }  
}
```

לאחר שליחת עדכון הדירוג, מנהל הליגה שולח הודעה ROUND_COMPLETED לסמן את סיום המഴור:

סיום מഴור - ROUND_COMPLETED

```
{  
  "jsonrpc": "2.0",  
  "method": "notify_round_completed",  
  "params": {  
    "protocol": "league.v2",  
    "message_type": "ROUND_COMPLETED",  
    "sender": "league_manager",  
    "timestamp": "2025-01-15T10:20:05Z",  
    "conversation_id": "conv-round-1-complete",  
    "league_id": "league_2025_even_odd",  
    "round_id": 1,  
    "matches_played": 2,  
    "next_round_id": 2  
  },  
  "id": 1402  
}
```

8.9 שלב 7 : סיום הליגה

לאחר סיום כל המחזוריים, מנהל הליגה שולח הודעה LEAGUE_COMPLETED

סיכום ליגה – LEAGUE_COMPLETED

```
{  
  "jsonrpc": "2.0",  
  "method": "notify_league_completed",  
  "params": {  
    "protocol": "league.v2",  
    "message_type": "LEAGUE_COMPLETED",  
    "sender": "league_manager",  
    "timestamp": "2025-01-15T12:00:00Z",  
    "conversation_id": "conv-league-complete",  
    "league_id": "league_2025_even_odd",  
    "total_rounds": 3,  
    "total_matches": 6,  
    "champion": {  
      "player_id": "P01",  
      "display_name": "AgentAlpha",  
      "points": 7  
    },  
    "final_standings": [  
      {"rank": 1, "player_id": "P01", "points": 7},  
      {"rank": 2, "player_id": "P03", "points": 5},  
      {"rank": 3, "player_id": "P04", "points": 4},  
      {"rank": 4, "player_id": "P02", "points": 2}  
    ]  
  },  
  "id": 2001  
}
```

8.10 טיפול בשגיאות

כאשר מתרחשת שגיאה, מנהל הליגה או השופט שולחים הודעה שגיאה מתאימה.

LEAGUE_ERROR – שגיאות אימות

```
{  
  "jsonrpc": "2.0",  
  "result": {  
    "protocol": "league.v2",  
    "message_type": "LEAGUE_ERROR",  
    "sender": "league_manager",  
    "timestamp": "2025-01-15T10:05:30Z",  
    "conversation_id": "conv-error-001",  
    "error_code": "E012",  
    "error_description": "AUTH_TOKEN_INVALID",  
    "context": {  
      "provided_token": "tok-invalid-xxx",  
      "action": "LEAGUE_QUERY"  
    }  
  },  
  "id": 1502  
}
```

שגיאת משחק – GAME_ERROR

```
{  
  "jsonrpc": "2.0",  
  "method": "notify_game_error",  
  "params": {  
    "protocol": "league.v2",  
    "message_type": "GAME_ERROR",  
    "sender": "referee:REF01",  
    "timestamp": "2025-01-15T10:16:00Z",  
    "conversation_id": "conv-r1m1-001",  
    "match_id": "R1M1",  
    "error_code": "E001",  
    "error_description": "TIMEOUT_ERROR",  
    "affected_player": "P02",  
    "action_required": "CHOOSE_PARITY_RESPONSE",  
    "retry_count": 0,  
    "max_retries": 3,  
    "consequence": "Technical loss if no response after retries"  
  },  
  "id": 1103  
}
```

8.11 כלים שאילתות זמינים

המסמך מגדיר כלים MCP גנריים שכלי סוכן יכול לחושף לצרכי דיבוג ובירור.

8.11.1 שאלת דירוג מנהל הליגה

שחקן שורצוה לוודא את דירוגו קורא למנהל הליגה:

שאילתת דירוג – LEAGUE_QUERY

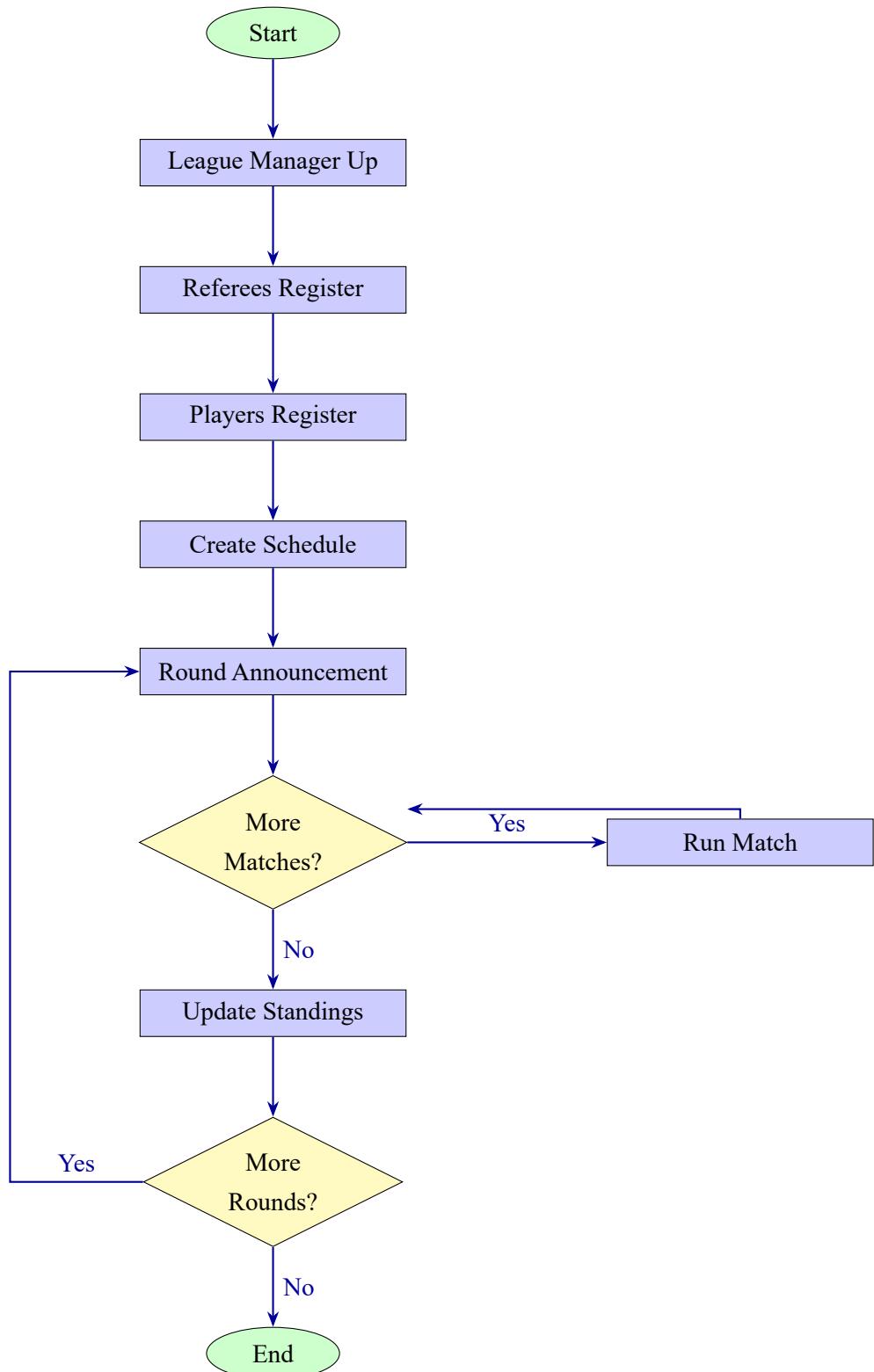
```
{  
  "jsonrpc": "2.0",  
  "method": "league_query",  
  "params": {  
    "protocol": "league.v2",  
    "message_type": "LEAGUE_QUERY",  
    "sender": "player:P01",  
    "timestamp": "2025-01-15T10:25:00Z",  
    "conversation_id": "conv-query-standings-001",  
    "auth_token": "tok-p01-xyz789",  
    "league_id": "league_2025_even_odd",  
    "query_type": "GET_STANDINGS"  
  },  
  "id": 1501  
}
```

מנהל הליגה מחזיר מחזיר שכלל result standings אובייקט זהה LEAGUE_STANDINGS_UPDATE.

8.11.2 כלים נוספים

- **כלי במנהל הליגה:** get_standings – מחזיר את מצב הטבלה העדכני.
- **כלי בשופט:** get_match_state – מחזיר מצב משחק קיים (לצרכי דיבוב).
- **כלי בשחקן:** get_player_state – נותן את ההיסטוריה המשחקים.

8.12 דיאגרמת זרימה מלאה



8.13 טבלת תפקידי הסוכנים

טבלה 17: תפקידי הסוכנים במערכת

סוכן	פורט	תפקיד בliga	מתקשר עם
League Manager	8000	Orchestrator ליגה, טבלה, מחזורים	שופטים ושחקנים
ahplA Referee	8001	Orchestrator משחקים	מנהל ליגה, שחקנים
ateB Referee	8002	Orchestrator משחקים	מנהל ליגה, שחקנים
P01 Player Agent	8101	בוחר שחקו, even/odd	שופט, מנהל ליגה
P02 Player Agent	8102	שחקן	שופט, מנהל ליגה
P03 Player Agent	8103	שחקן	שופט, מנהל ליגה
P04 Player Agent	8104	שחקן	שופט, מנהל ליגה

8.14 סיכום

נספח זה הציג את:

- **תצורת המערכת:** הקצתת פורטים וטרמינלים ל-7 סוכנים.
- **סדר הפעלה:** מנהל ליגה → שופטים → שחקנים.
- **זרימת הרישום:** LEAGUE_REGISTER ו REFEREE_REGISTER .auth_token
- **ניהול מחזור:** ROUND_COMPLETED ו ROUND_ANNOUNCEMENT
- **ניהול שחק:** GAME_OVER עד GAME_INVITATION
- **עדכון דירוג:** LEAGUE_STANDINGS_UPDATE ו MATCH_RESULT_REPORT
- **סיום ליגה:** LEAGUE_COMPLETED עם הכרזת אלוף.
- **טיפול בשגיאות:** GAME_ERROR ו LEAGUE_ERROR
- **שאילות:** LEAGUE_QUERY לקבלת מידע מעודכן.

כל התקשרות מתבצעת באמצעות JSON-RPC 2.0. כל ההודעות כוללות מעתפת (Envelope) אחת עם שדות חובה: timestamp ,sender ,message_type ,protocol (באזור זמן UTC) ,conversation_id ו-id. האורקסטרטורים (מנהל ליגה ושופטים) מנהלים את זרימת ההודעות בכל רגע.

9 פרוטוקול נתוני הליגה

9.1 מבוא: הקוד הגנטי של חברת הסוכנים

כasher anu bonyim chibra shel socnims ottonomim - shaknims, shofetim v'manhali ligah - anu le'ma'aseh yozrim terbrot digitalit chadsha. cmu b'khol chibra anoushit, g'm ca'an n'drashim shelosha y'sodot kritiyim:

1. **חוקים משותפים** - ה프וטוקול שהגדנו בפרקים הקודמים.
2. **זיכרון קולקטיבי** - היכולת לשמר ולשחזר מידע היסטורי.
3. **קוד גנטי** - הkonfiguracija שמנדרה את ה-DNA של כל סוכן.

נספח זה מתאר את "בסיס הנתונים על קובצי JSON" - ארכיטקטורת שלוש שכבות שמאפשרת למערכת לצמוח לקנה מידה של אלפי סוכנים וליגות.

9.2 ארכיטקטורת שלוש השכבות



9.2.1 עקרונות ניהול

כל קובץ במערכת עומד בעקרונות הבאים:

- **מזהה ייחודי (id)**: כל אובייקט ראשי מקבל מזהה חד-ערכי.
- **גרסת סכמה (schema_version)**: מאפשר מיגרציות עתידיות.
- **חותמת זמן (last_updated)**: בפורמט UTC/ISO-8601.
- **תאימות ל프וטוקול**: כל השדות תואמים ל-2.v2.league.

9.3 שכבה הקונפיגורציה - config/

שכבה זו מכילה את ה"קוד הגנטי" של המערכת - הגדירות סטטיות שנקראות בעליית הסוכנים.

9.3.1 קובץ מערכת גלובלי - config/system.json

- **מטרה:** פרמטרים גלובליים לכל המערכת.
 - **משתמשים:** כל הסוכנים, Orchestrator עליון.
 - **מיקום:** SHARED/config/system.json
- קובץ זה מגדיר את ערכי בירית המוחדר עבורה:
- הגדרות רשת (network) – פורטים וכתובות.
 - הגדרות אבטחה (security) – טוקנים ו-TTL.
 - זמני המתנה (timeouts) – תואמים להגדרות ה프וטוקול בפרק 2.
 - מדיניות ניסיון חזר (retry_policy) – תואמת להגדרות הפרוטוקול.

דוגמה: מבנה system.json (קטע)

```
{  
  "schema_version": "1.0.0",  
  "system_id": "league_system_prod",  
  "protocol_version": "league.v2",  
  "timeouts": {  
    "move_timeout_sec": 30,  
    "generic_response_timeout_sec": 10  
  },  
  "retry_policy": {  
    "max_retries": 3,  
    "backoff_strategy": "exponential"  
  }  
}
```

9.3.2 רישום סוכנים - config/agents/agents_config.json

- **מטרה:** ניהול מרכזי של אלף סוכנים.
 - **משתמשים:** מנהל הליגה, כל Deployment.
 - **מיקום:** SHARED/config/agents/agents_config.json
- קובץ זה מכיל את "ספר האזרכיס" של חברת הסוכנים:
- league_manager – פרטיה מנהל הליגה.
 - referees[] – רשימת כל השופטים הרשומים.
 - players[] – רשימת כל השחקנים הרשומים.

9.3.3 **קונפיגורציה ליגה - config/leagues/<league_id>.json**

- **מטרה:** הגדרות ספציפיות לliga.
- **משתמשים:** מנהל הליגה, שופטים.
- **מיקום:** SHARED/config/leagues/league_2025_even_odd.json
- כל ליגה היא "מדינה" עצמאית עם חוקים משלها:

דוגמה: קונפיגורציה ליגה (קטע)

```
{  
  "league_id": "league_2025_even_odd",  
  "game_type": "even_odd",  
  "status": "ACTIVE",  
  "scoring": {  
    "win_points": 3,  
    "draw_points": 1,  
    "loss_points": 0  
  },  
  "participants": {  
    "min_players": 2,  
    "max_players": 10000  
  }  
}
```

9.3.4 **רישום סוגי משחקים - config/games/games_registry.json**

- **מטרה:** רישום כל סוגי המשחקים הנתמכים.
- **משתמשים:** שופטים (לטינית מודול חוקים), מנהל ליגה.
- **מיקום:** SHARED/config/games/games_registry.json
- המערכת תומכת בסוגי משחקים רבים. כל משחק מגדיר:
 - **game_type** – מזאה ייחודי.
 - **rules_module** – מודול החוקים לטעינה.
 - **max_round_time_sec** – זמן מקסימלי לסיוב.

9.3.5 **ברירות מחדל לסוכנים - config/defaults/**

- **מטרה:** ערכי ברירת מחדל לפי סוג סוכן.
- **קבצים:** player.json, referee.json
- **מיקום:** SHARED/config/defaults/
- קבצים אלה מאפשרים לסוכן חדש להתחיל לפעול עם הגדרות סבירות מוביל להגדיר כל פרמטר בנפרד.

9.4 שכבת נתונים הריצה - data/

אם שכבת הkonfiguraciah היא ה"קוד המקורי", שכבת נתונים הריצה היא ה"איךرون ההיסטורי" של החברה. כאן נשמרים כל האירועים שקרוים במערכת.

9.4.1 טבלת דירוג - data/leagues/<league_id>/standings.json

- **מטרה:** מצב הדירוג העדכני של הליגה.
- **מעדכן:** מנהל הליגה (אחרי MATCH_RESULT_REPORT).
- **מיקום:** SHARED/data/leagues/league_2025_even_odd/standings.json

דוגמה: מבנה טבלת דירוג

```
{  
  "schema_version": "1.0.0",  
  "league_id": "league_2025_even_odd",  
  "version": 12,  
  "rounds_completed": 3,  
  "standings": [  
    {  
      "rank": 1,  
      "player_id": "P01",  
      "display_name": "Agent_Alpha",  
      "wins": 4, "draws": 1, "losses": 1,  
      "points": 13  
    }  
  ]  
}
```

9.4.2 היסטוריה מוחזרים - data/leagues/<league_id>/rounds.json

- **מטרה:** תיעוד כל המוחזרים שהתקיימו.
- **מעדכן:** מנהל הליגה (אחרי ROUND_COMPLETED).
- **מיקום:** SHARED/data/leagues/league_2025_even_odd/rounds.json

9.4.3 נתונים משחק בודד - data/matches/<league_id>/<match_id>.json

- **מטרה:** תיעוד מלא של משחק בודד.
- **מעדכן:** השופט שניהל את המשחק.
- **מיקום:** SHARED/data/matches/league_2025_even_odd/R1M1.json
 - קובץ זה הוא ה"תעודת זהות" של המשחק ומכיל:
 - מצב המשחק וזמן.
 - lifecycle

- transcript[] – כל ההודעות שהוחלפו (ההיסטוריה ממלחכים).
- .(GAME_OVER – התוצאה הסופית (תואם ל- result -

9.4.4 היסטוריה שחון – data/players/<player_id>/history.json

- מטרה: "אזכור אישי" של השחקן.
 - משתמש: השחקן עצמו (לבנית אסטרטגיה).
 - מיקום: SHARED/data/players/P01/history.json
- שחקן חכם יכול להשתמש בקובץ זה כ"אזכור" לשיפור האסטרטגיה שלו:

דוגמה: היסטוריה שחון

```
{
  "player_id": "P01",
  "stats": {
    "total_matches": 20,
    "wins": 12, "losses": 5, "draws": 3
  },
  "matches": [
    {
      "match_id": "R1M1",
      "opponent_id": "P02",
      "result": "WIN",
      "my_choice": "even",
      "opponent_choice": "odd"
    }
  ]
}
```

9.5 שכבת הלוגים – logs/

שכבה זו היא "מערכת העצבים" של החברה – היא מאפשרת לנו לראות מה באמת קורה במערכת המבוזרת.

9.5.1 לוג ליגה מרכז – logs/league/<league_id>/league.log.jsonl

- **פורמט:** JSON Lines (כל שורה אובייקט JSON נפרד).
- **משתמשים:** DevOps, תומכה טכנית.
- **מיקום:** SHARED/logs/league/league_2025_even_odd/league.log.jsonl

דוגמה: רשומת לוג ליגה

```
{  
  "timestamp": "2025-01-15T10:15:00Z",  
  "component": "league_manager",  
  "event_type": "ROUND_ANNOUNCEMENT_SENT",  
  "level": "INFO",  
  "details": {"round_id": 1, "matches_count": 2}  
}
```

9.5.2 לוג סוכן – logs/agents/<agent_id>.log.jsonl

- **מטרה:** מעקב פר-סוכן לדיבוג.
 - **משתמשים:** מפתחי הסוכן.
 - **מיקום:** SHARED/logs/agents/P01.log.jsonl
- כל סוכן מתעד את ההודעות שהוא שולח ומקבל, מה שמאפשר End-to-End Trace של כל אינטראקציה במערכת.

9.6 טבלת סיכום קבצים

9.7 שימוש בקבצים המשותפים

כל קובצי הדוגמה המתוארים בסוף זה זמינים בתיקייה המשותפת:

L07 / SHARED /

סטודנטים מוזמנים להשתמש בקבצים אלה כבסיס למימוש הסוכנים שלהם. הקבצים כוללים:

- דוגמאות מלאות לכל סוג קובץ.
- נתונים תואמים לפרטוקול league.v2.
- מבנה תיקיות מומלץ לפרויקט.

טבלה 18: סיכום קובצי הקונפיגורציה והנתונים

משתמש	מטרה	נתיב	שכבה
כל הסוכנים	פרמטרים גלובליים	config/system.json	קונפיג
מנהל ליגה	רישום סוכנים	config/agents/	קונפיג
מנהל ליגה	הדרות ליגה	config/leagues/	קונפיג
שופטים	רישום משחקים	config/games/	קונפיג
סוכנים	ברירות מחדל	config/defaults/	קונפיג
colm	טבלת דירוג	data/.../standings.json	ריצה
מנהל ליגה	ההיסטוריה מחזוריים	data/.../rounds.json	ריצה
אנלטיקה	פרטי משחק	data/matches/	ריצה
שחקן	ההיסטוריה אישית	data/players/	ריצה
DevOps	לוג מרכזי	logs/league/	לוגים
פתחים	לוג סוכן	logs/agents/	לוגים

9.8 סיכום

ארכיטקטורת שלוש השכבות שהציגו – קונפיגורציה, נתוני ריצה, ולוגים – מספקת את התשתיות הנדרשת לבניית מערכת סוכנים בקנה מידה גדול.

כמו בחברה אנושית, גם כאן:

- **הكونפיגורציה** היא ה"חוקה" – הכללים היסודיים שכולם מכירים.
- **נתוני הריצה** הם ה"ארכיוון ההיסטורי" – האזכור הקולקטיבי.
- **elogims** הם ה"עיתונות" – תיעוד מה שקרה בזמןאמת.

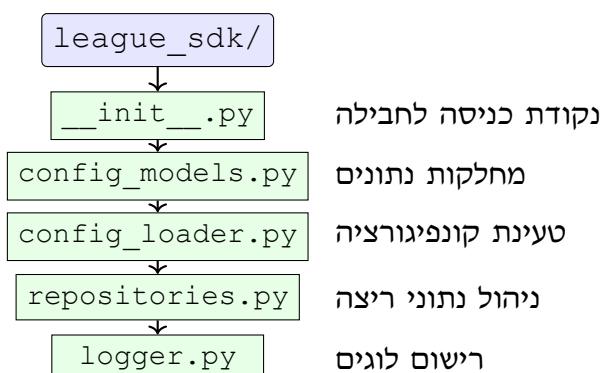
מבנה זה מכין את המערכת לצמיחה של אלפי סוכנים וליגות, תוך שמירה על סדר, עקביות, ויכולת מעקב.

10. ערכות כלים בפייתון

10.1 מבוא: מהקונפיגורציה לךוד

בנספח ב הציגו את ארכיטקטורת הנתונים מבוססת קובץ JSON – שלוש שכבות של קונפיגורציה, נתונים ריצה, ולוגים. אך כיצד סוכן בודד ניגש נתונים אלו בפועל? נספח זה מציג את league_sdk – ספרייה שմחברת בין קובץ ה-JSON לבין האובייקטים שבהם משתמשים הסוכנים. הספרייה מימושת שני דפוסי עיצוב מרכזים: 1. מודלים טיפוסיים (typed models) שמשקפים את מבנה ה-JSON. 2. שכבת הפשטה לגישה נתונים. Repository Pattern

10.2 מבנה הספרייה



10.3 מודלים טיפוסיים – config_models.py

10.3.1 הגישה: Dataclasses

Python 3.7+ מספקת את הדקטורו `@dataclass` שמאפשר הגדרת מחלקות נתונים בצורה תמציתית. כל שדה ב-JSON הופך לשדה במחלקה עם טיפוס מוגדר:

דוגמה: הגדרת Dataclass

```
from dataclasses import dataclass
from typing import List

@dataclass
class NetworkConfig:
    base_host: str
    default_league_manager_port: int
    default_referee_port_range: List[int]
    default_player_port_range: List[int]
```

10.3.2 מודלי קונפיגורציה מערכת

הקובץ מגדיר את כל המודלים התואימים ל-system.json

מודלי קונפיגורציה גלובלית

```
@dataclass
class SecurityConfig:
    enable_auth_tokens: bool
    token_length: int
    token_ttl_hours: int

@dataclass
class TimeoutsConfig:
    register_referee_timeout_sec: int
    register_player_timeout_sec: int
    game_join_ack_timeout_sec: int
    move_timeout_sec: int
    generic_response_timeout_sec: int

@dataclass
class SystemConfig:
    schema_version: str
    system_id: str
    protocol_version: str
    default_league_id: str
    network: NetworkConfig
    security: SecurityConfig
    timeouts: TimeoutsConfig
    # ...additional fields
```

10.3.3 מודלי סוכנים

כל סוג סוכן מקבל מחלוקת קונפיגורציה משלו:

מודלי קונפיגורציית סוכנים

```
@dataclass
class RefereeConfig:
    referee_id: str
    display_name: str
    endpoint: str
    version: str
    game_types: List[str]
    max_concurrent_matches: int
    active: bool = True

@dataclass
class PlayerConfig:
    player_id: str
    display_name: str
    version: str
    preferred_leagues: List[str]
    game_types: List[str]
    default_endpoint: str
    active: bool = True
```

10.3.4 מודלי ליגה

הגדירות ספציפיות לliga כוללות תזמון, ניקוד, ומשתתפים:

מודלי קונפיגורציית ליגה

```
@dataclass
class ScoringConfig:
    win_points: int
    draw_points: int
    loss_points: int
    technical_loss_points: int
    tiebreakers: List[str]

@dataclass
class LeagueConfig:
    schema_version: str
    league_id: str
    display_name: str
    game_type: str
    status: str
    scoring: ScoringConfig
    # ...additional fields
```

10.4 טווען קונפיגורציה – ConfigLoader

10.4.1 העיקרונות: טעינה עצלה עם מטמון

המחלקה ConfigLoader מיישמת את דפוס ה-Lazy Loading – קובצי הקונפיגורציה נתונים רק כשהם צריכים, ונשמרים במטמון לגישה חזרה:

מבנה ConfigLoader

```
class ConfigLoader:
    def __init__(self, root: Path = CONFIG_ROOT):
        self.root = root
        self._system = None          # lazy cache
        self._agents = None          # lazy cache
        self._leagues = {}           # league_id -> LeagueConfig

    def load_system(self) -> SystemConfig:
        """Load global system configuration."""
        if self._system:
            return self._system
        path = self.root / "system.json"
        data = json.loads(path.read_text(encoding="utf-8"))
        self._system = SystemConfig(...)
        return self._system
```

10.4.2 שיטות הטעינה

מספקת ממשק אחד לטעינת כל סוגי הקונפיגורציה ConfigLoader

טבלה 19: שיטות הטעינה של ConfigLoader

שיטה	מחזירה	תיאור
load_system()	SystemConfig	קונפיגורציה גלובלית
load_agents()	AgentsConfig	רשימת כל הסוכנים
load_league(id)	LeagueConfig	קונפיגורציה ליניה ספציפית
load_games_registry()	GamesRegistry	רישום סוגי המשחקים

10.4.3 שיטות עזר

בנוסף לטעינה ישירה, המחלקה מספקת שיטות נוחות לחיפוש:

שיטות עזר

```
def get_referee_by_id(self, referee_id: str) -> RefereeConfig:
    """Get a referee configuration by ID."""
    agents = self.load_agents()
    for ref in agents.referees:
        if ref.referee_id == referee_id:
            return ref
    raise ValueError(f"Referee not found: {referee_id}")

def get_player_by_id(self, player_id: str) -> PlayerConfig:
    """Get a player configuration by ID."""
    agents = self.load_agents()
    for player in agents.players:
        if player.player_id == player_id:
            return player
    raise ValueError(f"Player not found: {player_id}")
```

10.5 מאגרי נתונים – Repositories

10.5.1 דפוס המאגר (Repository Pattern)

בעוד ש-ConfigLoader מטפל בקונפיגורציה סטטית, שכבת המאגרים מטפלת בנתונים דינמיים. כל מאגר אחראי על קריאה, עדכון, ושמירה של סוג נתונים ספציפי.

מאגר טבלת דירוג

```

class StandingsRepository:
    def __init__(self, league_id: str, data_root: Path = DATA_ROOT):
        self.league_id = league_id
        self.path = data_root / "leagues" / league_id / "standings.json"
        self.path.parent.mkdir(parents=True, exist_ok=True)

    def load(self) -> Dict:
        """LoadStandingsFromJSONFile."""
        if not self.path.exists():
            return {"schema_version": "1.0.0", "standings": []}
        return json.loads(self.path.read_text(encoding="utf-8"))

    def save(self, standings: Dict) -> None:
        """SaveStandingsToJsonFile."""
        standings["last_updated"] = datetime.utcnow().isoformat() + "Z"
        self.path.write_text(json.dumps(standings, indent=2))

    def update_player(self, player_id: str, result: str, points: int):
        """UpdateAPlayer'sStandingsAfterAMatch."""
        standings = self.load()
        # ... update logic
        self.save(standings)

```

10.5.3 מאגרים נוספים

הספרייה כוללת מאגרים נוספים לניהול נתונים ריכזה:

טבלה 20: מאגרי נתונים זמינים

מאגר	קובץ	תפקיד
StandingsRepository	standings.json	טבלת דירוג ליגה
RoundsRepository	rounds.json	היסטורית מחזוריים
MatchRepository	<match_id>.json	נתוני משחק בודד
PlayerHistoryRepository	history.json	היסטורית שחקן

10.6 רישום לוגים – JsonLogger

10.6.1 פורמט JSON Lines

הספרייה משתמשת בפורמט JSON Lines (JSONL) – כל שורה בקובץ הלוג היא אובייקט JSON עצמאי. פורמט זה מאפשר:

- הוספת רשומות חדשות בצורה עילית (append-only).
- קראיה וניתוח באמצעות כלים סטנדרטיים.
- סטרימינג של לוגים בזמן אמת.

10.6.2 מחלקה הלוגר

מחלקה JsonLogger

```
class JsonLogger:  
    def __init__(self, component: str, league_id: str | None = None):  
        self.component = component  
        # Determine log directory  
        if league_id:  
            subdir = LOG_ROOT / "league" / league_id  
        else:  
            subdir = LOG_ROOT / "system"  
        subdir.mkdir(parents=True, exist_ok=True)  
        self.log_file = subdir / f"{component}.log.jsonl"  
  
    def log(self, event_type: str, level: str = "INFO", **details):  
        entry = {  
            "timestamp": datetime.utcnow().isoformat() + "Z",  
            "component": self.component,  
            "event_type": event_type,  
            "level": level,  
            **details,  
        }  
        with self.log_file.open("a", encoding="utf-8") as f:  
            f.write(json.dumps(entry, ensure_ascii=False) + "\n")
```

10.6.3 שיטות נוחות

הלוגר מספק שיטות לרמות לוג שונות ולאירועים נפוצים:

שיטות נוחות ללוגר

```
def debug(self, event_type: str, **details):
    self.log(event_type, level="DEBUG", **details)

def info(self, event_type: str, **details):
    self.log(event_type, level="INFO", **details)

def warning(self, event_type: str, **details):
    self.log(event_type, level="WARNING", **details)

def error(self, event_type: str, **details):
    self.log(event_type, level="ERROR", **details)

def log_message_sent(self, message_type: str, recipient: str, **details):
    self.debug("MESSAGE_SENT", message_type=message_type,
              recipient=recipient, **details)
```

10.7.1 דוגמה: מנהל ליגת

שימוש ב-ConfigLoader במנהל ליגת

```
from league_sdk import ConfigLoader, JsonLogger

class LeagueManager:
    def __init__(self, league_id: str):
        loader = ConfigLoader()
        self.system_cfg = loader.load_system()
        self.agents_cfg = loader.load_agents()
        self.league_cfg = loader.load_league(league_id)

        self.logger = JsonLogger("league_manager", league_id)

        # Build lookup maps
        self.referees_by_id = {
            r.referee_id: r.endpoint
            for r in self.agents_cfg.referees if r.active
        }

    def get_timeout_for_move(self) -> int:
        return self.system_cfg.timeouts.move_timeout_sec
```

שימוש ב-ConfigLoader בשופט

```

from league_sdk import ConfigLoader, JsonLogger

class RefereeAgent:
    def __init__(self, referee_id: str, league_id: str):
        loader = ConfigLoader()
        self.system_cfg = loader.load_system()
        self.league_cfg = loader.load_league(league_id)
        self.self_cfg = loader.get_referee_by_id(referee_id)

        self.logger = JsonLogger(f"referee:{referee_id}", league_id)

    def register_to_league(self):
        payload = {
            "jsonrpc": "2.0",
            "method": "register_referee",
            "params": {
                "protocol": self.system_cfg.protocol_version,
                "message_type": "REFEREE_REGISTER_REQUEST",
                "referee_meta": {
                    "display_name": self.self_cfg.display_name,
                    "version": self.self_cfg.version,
                    "game_types": self.self_cfg.game_types,
                }
            }
        }
        #
        # ... send request

```

רישום שגיאת TIMEOUT

```
logger = JsonLogger("referee:REF01", "league_2025_even_odd")

logger.error(
    "GAME_ERROR",
    match_id="R1M1",
    error_code="TIMEOUT_MOVE",
    player_id="P02",
    timeout_sec=30,
)
# Output to logs/league/league_2025_even_odd/referee_REF01.log.jsonl:
# {"timestamp": "2025-01-15T10:15:00Z", "component": "referee:REF01",
# "event_type": "GAME_ERROR", "level": "ERROR", "match_id": "R1M1", ...}
```

10.8 סיכום

ספריית league_sdk מספקת שכבה פשוטה נקייה בין קובצי ה-JSON לקוד הסוכנים:

- config_models.py – מגדיר טיפוסים בטוחים (type-safe) לכל מבנה נתונים.
- config_loader.py – מספק גישה נוחה לקונפיגורציה עם מטמון.
- repositories.py – מנהל נתונים ריצה בדף Repository.
- logger.py – מאפשר רישום לוגים מובנה בפורמט JSON.

שימוש בספרייה זו מבטיח:

1. **עקביות** – כל הסוכנים משתמשים באותם מודלים נתונים.
2. **תחזוקתיות** – שינויים במבנה הנתונים מתרכזים במקום אחד.
3. **בטיחות טיפוסים** – שגיאות נתפסות בזמן כתיבת הקוד.
4. **יכולת ניפוי** – לוגים מובנים מאפשרים מעקב קל אחר בעיות.

הספרייה זמינה בתיקייה:

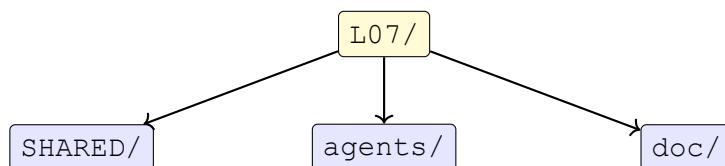
L07/SHARED/league_sdk/

11 מבנה הפרויקט

11.1 מבוא: מפת הדרכים

לאחר שהכרנו את ה프וטוקול, הودעות ה-JSON, ארכיטקטורת הנתונים, וספריית ה-*Python* – הגיע הזמן לראות את התמונה המלאה. נספח זה מציג את מבנה התיקיות והקבצים המומלץ לפרויקט הליגה, כך שכל סטודנט יוכל להתחיל לעבוד עם בסיס מסודר וモוכן.

11.2 עץ התיקיות הראשי



טבלה 21: תיקיות הבסיס של הפרויקט

תיקיה	תיאור
SHARED/	משאבים משותפים – קונפיגורציה, נתונים, לוגים, וספריית SDK
agents/	קוד הסוכנים – כל סוכן בתיקייה נפרד
doc/	תיעוד הפרויקט – מסמכים ומפרטים

11.3 תיקיית המשאבים המשותפים – /SHARED/

תיקיה זו מכילה את כל המשאבים המשותפים לכל הסוכנים במערכת.

SHARED/

```
└── config/                                # Configuration layer
    └── system.json                         # Global system settings
    └── agents/
        └── agents_config.json      # All agents registry
    └── leagues/
        └── league_2025_even_odd.json
    └── games/
        └── games_registry.json    # Supported game types
    └── defaults/
        └── referee.json          # Default referee settings
```

```

    └── player.json          # Default player settings

└── data/                  # Runtime data layer
    ├── leagues/
    │   └── league_2025_even_odd/
    │       ├── standings.json    # Current standings
    │       └── rounds.json      # Round history
    ├── matches/
    │   └── league_2025_even_odd/
    │       ├── R1M1.json        # Match R1M1 data
    │       └── R1M2.json        # Match R1M2 data
    └── players/
        ├── P01/
        │   └── history.json     # P01 match history
        └── P02/
            └── history.json     # P02 match history

└── logs/                  # Logging layer
    ├── league/
    │   └── league_2025_even_odd/
    │       └── league.log.jsonl
    ├── agents/
    │   ├── REF01.log.jsonl
    │   ├── P01.log.jsonl
    │   └── P02.log.jsonl
    └── system/
        └── orchestrator.log.jsonl

└── league_sdk/            # Python SDK
    ├── __init__.py
    ├── config_models.py      # Dataclass definitions
    ├── config_loader.py      # ConfigLoader class
    ├── repositories.py       # Data repositories
    └── logger.py             # JsonLogger class

```

11.4 תקיות הסוכנים - agents/

כל סוכן חי בתיקייה נפרדת עם מבנה אחיד:

```
agents/
  └── league_manager/
      ├── main.py                      # Entry point
      ├── handlers.py                  # Message handlers
      ├── scheduler.py                # Round scheduling
      └── requirements.txt

  └── referee_REF01/
      ├── main.py                      # Entry point
      ├── game_logic.py               # Even/Odd rules
      ├── handlers.py                  # Message handlers
      └── requirements.txt

  └── player_P01/
      ├── main.py                      # Entry point
      ├── strategy.py                 # Playing strategy
      ├── handlers.py                  # Message handlers
      └── requirements.txt

  └── player_P02/
      ├── main.py
      ├── strategy.py
      ├── handlers.py
      └── requirements.txt
```

11.4.1 מבנה סוכן טיפוסי

כל סוכן מכיל את הקבצים הבאים:

11.5 תקיות התיעוד - doc/

```
doc/
  └── protocol-spec.md            # Protocol specification
  └── message-examples/
      └── registration/
          ├── referee_register_request.json
          └── player_register_request.json
```

טבלה 22: קבצי סוכן טיפוסי

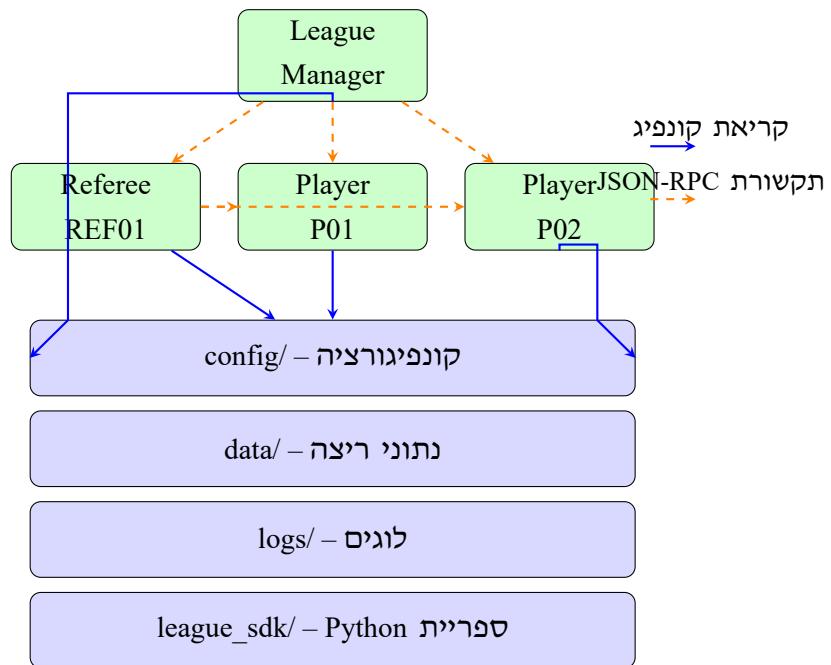
קובץ	תפקיד
main.py	נקודות כניסה - אתחול השירות וטעינה קונפיגורציה
handlers.py	טיפול בהודעות כניסה לפי סוג
strategy.py	לוגיקת קבלת החלטות (לשחקנים)
game_logic.py	חוקי המשחק (לשופטים)
requirements.txt	תלויות Python

```

|   └── game-flow/
|       ├── game_start.json
|       ├── move_request.json
|       └── game_over.json
|   └── errors/
|       ├── timeout_error.json
|       └── invalid_move.json
└── diagrams/
    ├── architecture.png
    └── message-flow.png

```

11.6 דיאגרמת ארכיטקטורה



11.7 זרימת נתונים

11.7.1 קרייה וכתיבה

טבלה 23: הרשותות גישה לקבצים

קובץ/תיקייה	קורא	כותב	הערות
config/*	cols	מנהל מערכת	בלבד קרייה לסוכנים
standings.json	cols	מנהל ליגה	עדכו משחיק
rounds.json	cols	מנהל ליגה	ההיסטוריה מחזוריים
matches/*.json	cols	שופט	קובץ למשחק
history.json	שחקן	שחקן	ההיסטוריה אישית
logs/*	DevOps	כל סוכן	לוג משלו בלבד

11.8 התקנה והפעלה

11.8.1 דרישות מקדימות

Python 3.10+ -
pip או poetry לניהול חבילות -
גישה לתקינות/SHARED -

11.8.2 התקנת תלויות

```
# Install league_sdk
cd SHARED
pip install -e league_sdk/

# Install agent dependencies
cd ../agents/player_P01
pip install -r requirements.txt
```

11.8.3 הפעלת סוכן

```
# Start League Manager
cd agents/league_manager
python main.py --league-id league_2025_even_odd

# Start Referee
cd agents/referee_REF01
python main.py --referee-id REF01 \
               --league-id league_2025_even_odd

# Start Player
cd agents/player_P01
python main.py --player-id P01 \
               --league-id league_2025_even_odd
```

11.9 רשימת קבצים מלאה

להלן רשימה מלאה של כל הקבצים בפרויקט:

```
L07/
├─ SHARED/
```

```

    config/
        system.json
        agents/agents_config.json
        leagues/league_2025_even_odd.json
        games/games_registry.json
        defaults/{referee,player}.json
    data/
        leagues/<league_id>/{standings,rounds}.json
        matches/<league_id>/<match_id>.json
        players/<player_id>/history.json
    logs/
        league/<league_id>/*.log.jsonl
        agents/*.log.jsonl
        system/*.log.jsonl
    league_sdk/
        __init__.py
        config_models.py
        config_loader.py
        repositories.py
        logger.py
agents/
    league_manager/{main,handlers,scheduler}.py
    referee_REF01/{main,handlers,game_logic}.py
    player_*/{main,handlers,strategy}.py
doc/
    protocol-spec.md
    message-examples/**/*.json

```

11.10 סיכום

מבנה הפרויקט המוצג בנספח זה מספק:

1. **הפרזה ברורה** – כל רכיב בתיקייה משלה.
2. **משאבים משותפים** – /SHARED מרכז את כל הנתונים.
3. **עכמאות סוכנים** – כל סוכן יכול לפעול באופן עצמאי.
4. **תיעוד מובנה** – דוגמאות ומפורטים ב-/doc.
5. **יכולת הרחבה** – קל להווסף סוכנים וליגות חדשות.

הקבצים המלאים זמינים בתיקייה:

L07/SHARED/