

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 MCP – Model Context Protocol
12	1.2.1 עקרונות הפרוטוקול
13	1.2.2 ארכיטקטורת Host/Server
13	1.3 תחבורת HTTP על localhost
13	1.3.1 הגדרת פורטים
13	1.3.2 דוגמה לכתובת סוכן
14	1.4 מבנה הודעת 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 לשלוש 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 אזור 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: יצירת לוח משחקים
20	2.5.4 שלב 4: הכרזה על מחזור

21	2.5.5	שלב 5: ניהול משחקים
21	2.5.6	שלב 6: עדכון דירוג
21	2.6	דיאגרמת זרימה כללית
21	2.7	זמני תגובה – Timeouts
22	2.8	מחזור חיים של סוכן – Agent Lifecycle
22	2.8.1	מצבי הסוכן
22	2.8.2	דיאגרמת 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	קודי שגיאה נפוצים
24	2.9.4	מדיניות ניסיון חוזר – 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 יחיד
25	2.11.2	תקשורת דרך Orchestrator
25	2.11.3	טיפול בכשלים
26	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 בחירות
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	3.3.4	מצב FINISHED
28	3.4	שיטת הניקוד
28	3.4.1	ניקוד למשחק
28	3.4.2	דירוג בליגה
28	3.5	ליגת Round-Robin
28	3.5.1	מספר משחקים
29	3.5.2	לוח משחקים לדוגמה
29	3.6	אסטרטגיות לשחקנים
29	3.6.1	אסטרטגיה אקראית
29	3.6.2	אסטרטגיה מבוססת היסטוריה
29	3.6.3	אסטרטגיה מונחית LLM
30	3.7	מודול חוקי המשחק
30	3.7.1	ממשק המודול
30	3.7.2	יתרון ההפרדה
30	3.8	הרחבה למשחקים נוספים
30	3.8.1	הפשטת מהלך - GAME_MOVE
31	3.8.2	מבנה הודעת מהלך גנרית
32	3.8.3	רישום סוגי משחקים - Game Registry
32	3.8.4	יתרונות ההפשטה
33		4	מבני הודעות JSON
33	4.1	הודעות רישום שופט לליגה
33	4.1.1	בקשת רישום שופט - REFEREE_REGISTER_REQUEST
34	4.1.2	תגובת רישום שופט - REFEREE_REGISTER_RESPONSE
35	4.2	הודעות רישום שחקן לליגה
35	4.2.1	בקשת רישום שחקן - LEAGUE_REGISTER_REQUEST
36	4.2.2	תגובת רישום - LEAGUE_REGISTER_RESPONSE
37	4.3	הודעות מחזור
37	4.3.1	הכרזת מחזור - 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 בחירה במשחק זוגי/אי-זוגי
39	4.5.1	בקשת בחירה - CHOOSE_PARITY_CALL
39	4.5.2	תגובת בחירה - CHOOSE_PARITY_RESPONSE

40	הודעות תוצאה	4.6
40	GAME_OVER – סיום משחק	4.6.1
41	MATCH_RESULT_REPORT – דיווח תוצאה לליגה	4.6.2
41	הודעות דירוג	4.7
42	LEAGUE_STANDINGS_UPDATE – עדכון דירוג	4.7.1
42	הודעות סיום מחזור וליגה	4.8
43	ROUND_COMPLETED – סיום מחזור	4.8.1
44	LEAGUE_COMPLETED – סיום ליגה	4.8.2
44	הודעות שאילתא	4.9
45	LEAGUE_QUERY – שאילתת ליגה	4.9.1
46	LEAGUE_QUERY_RESPONSE – תגובת שאילתה	4.9.2
46	הודעות שגיאה	4.10
47	LEAGUE_ERROR – שגיאה ברמת הליגה	4.10.1
48	GAME_ERROR – שגיאה ברמת המשחק	4.10.2
48	טבלת סיכום הודעות	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

5 מדריך מימוש

51	ארכיטקטורה 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 שחקן	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 לליגה	5.4.2
55	לוגיקת קביעת מנצח	5.4.3
55	מימוש מנהל ליגה	5.5
55	כלים נדרשים	5.5.1

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

70	6.4.1	שלב 1: פיתוח מקומי
70	6.4.2	שלב 2: ליגה פרטית
71	6.4.3	שלב 3: בדיקת תאימות עם סטודנטים אחרים
71	6.4.4	מבט לעתיד: ליגת כיתה
71	6.5	הגשה
71	6.5.1	קבצים להגשה
71	6.5.2	פורמט הגשה
71	6.6	דגשים כלליים לבדיקת העבודה
72	6.7	שאלות נפוצות
72	6.7.1	האם אפשר להשתמש בספריות חיצוניות?
72	6.7.2	האם חייבים להשתמש ב-Python?
72	6.7.3	מה קורה אם הסוכן שלי קורס?
72	6.7.4	האם אפשר לעדכן את הסוכן אחרי ההגשה?
72	6.7.5	איך אדע מה הדירוג שלי?
73	6.8	סיכום
74	7	למידת MCP דרך תרגיל הליגה
74	7.1	השחקן כסוכן AI
74	7.1.1	האם סוכן השחקן הוא סוכן AI?
74	7.1.2	ארבעת המאפיינים של סוכן AI
74	7.2	השחקן בארכיטקטורת MCP
75	7.2.1	שרת או לקוח?
75	7.2.2	היחסים מול השופט ומנהל הליגה
75	7.3	השופט ומנהל הליגה כסוכני AI
75	7.3.1	סוכנים בדרגה גבוהה
75	7.3.2	שרתי MCP שפועלים גם כלקוחות
76	7.4	היפוך התפקידים: תובנה מרכזית
76	7.4.1	הפרדיגמה המסורתית
76	7.4.2	היפוך התפקידים בליגה
77	7.5	עקרון הפרדת השכבות
77	7.5.1	שלוש שכבות נפרדות
77	7.5.2	היתרון של ההפרדה
78	7.6	תפקיד ה-LLM בסוכן השרת
78	7.6.1	הדילמה
78	7.6.2	הפתרון: הפרדת תפקידים

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

96	8.8	שלב 6: סיום מחזור ועדכון דירוג
99	8.9	שלב 7: סיום הליגה
99	8.10	טיפול בשגיאות
100	8.10.1	שגיאת אימות
101	8.10.2	שגיאת משחק – זמן תגובה
101	8.11	כלי שאילתות זמינים
102	8.11.1	שאילתת דירוג ממנהל הליגה
102	8.11.2	כלים נוספים
103	8.12	דיאגרמת זרימה מלאה
104	8.13	טבלת תפקידי הסוכנים
104	8.14	סיכום
105	9	פרוטוקול נתוני הליגה
105	9.1	מבוא: הקוד הגנטי של חברת הסוכנים
105	9.2	ארכיטקטורת שלוש השכבות
105	9.2.1	עקרונות מנחים
105	9.3	שכבת הקונפיגורציה – config/
106	9.3.1	קובץ מערכת גלובלי – config/system.json
106	9.3.2	רישום סוכנים – config/agents/agents_config.json
107	9.3.3	קונפיגורציית ליגה – config/leagues/<league_id>.json
107	9.3.4	רישום סוגי משחקים – config/games/games_registry.json
107	9.3.5	ברירות מחדל לסוכנים – config/defaults/
108	9.4	שכבת נתוני הריצה – data/
108	9.4.1	טבלת דירוג – data/leagues/<league_id>/standings.json
108	9.4.2	היסטוריית מחזורים – data/leagues/<league_id>/rounds.json
108	9.4.3	נתוני משחק בודד – data/matches/<league_id>/<match_id>.json
109	9.4.4	היסטוריית שחקן – data/players/<player_id>/history.json
109	9.5	שכבת הלוגים – logs/
110	9.5.1	לוג ליגה מרכזי – logs/league/<league_id>/league.log.jsonl
110	9.5.2	לוג סוכן – logs/agents/<agent_id>.log.jsonl
110	9.6	טבלת סיכום קבצים
110	9.7	שימוש בקבצים המשותפים
112	9.8	סיכום
113	10	ערכת כלים בפייתון

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	העיקרון: 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 הדרכים	11.1
125	עץ התיקיות הראשי	11.2
125	תיקיית המשאבים המשותפים – SHARED/	11.3
127	תיקיית הסוכנים – agents/	11.4
127	מבנה סוכן טיפוסי	11.4.1
127	תיקיית התיעוד – doc/	11.5
129	דיאגרמת ארכיטקטורה	11.6
129	זרימת נתונים	11.7
129	קריאה וכתיבה	11.7.1

130	התקנה והפעלה	11.8
130	דרישות מקדימות	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

12 מקורות 133

1 מבוא: סוכני AI ופרוטוקול MCP

1.1 מהו סוכן AI?

סוכן AI הוא תוכנה אוטונומית. הסוכן מקבל מידע מהסביבה. הוא מעבד את המידע. לאחר מכן הוא מבצע פעולות.

סוכן AI שונה מתוכנית רגילה. תוכנית רגילה מבצעת הוראות קבועות מראש. סוכן AI מחליט בעצמו מה לעשות. ההחלטה מבוססת על המצב הנוכחי.

1.1.1 מאפיינים של סוכן AI

לכל סוכן AI יש מספר מאפיינים:

- **אוטונומיות** – הסוכן פועל באופן עצמאי.
- **תפיסה** – הסוכן קולט מידע מהסביבה.
- **פעולה** – הסוכן משפיע על הסביבה.
- **תכליתיות** – לסוכן יש מטרה מוגדרת.

בספרו של ד"ר יורם סגל "סוכני AI עם MCP" [1], מוסבר כיצד סוכנים מתקשרים. הספר מציג את פרוטוקול MCP בהרחבה. אנו נשתמש בעקרונות אלה בתרגיל.

1.2 פרוטוקול MCP – Model Context Protocol

MCP הוא פרוטוקול תקשורת. הפרוטוקול פותח על ידי חברת Anthropic. הוא מאפשר לסוכני AI לתקשר זה עם זה.

1.2.1 עקרונות הפרוטוקול

הפרוטוקול מבוסס על מספר עקרונות:

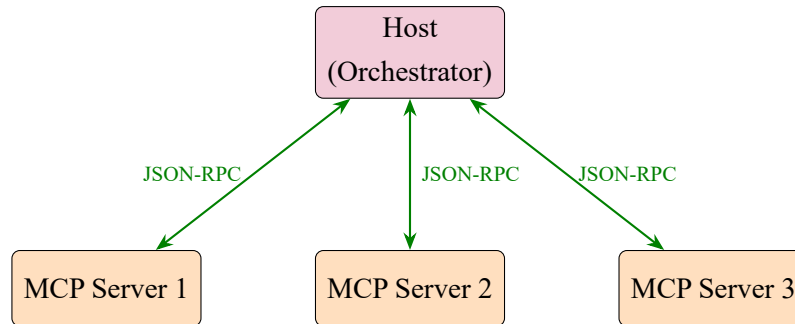
1. **הודעות מובנות** – כל הודעה היא אובייקט JSON.
2. **תקן JSON-RPC 2.0** – הפרוטוקול משתמש בתקן זה.
3. **כלים (sloot)** – סוכנים חושפים פונקציות כ"כלים".
4. **תחבורה גמישה** – אפשר להשתמש ב-HTTP או stdio.

1.2.2 ארכיטקטורת Host/Server

במערכת MCP יש שני סוגי רכיבים:

שרת MCP – רכיב שמספק שירותים. השרת חושף "כלים" שאפשר לקרוא להם. כל כלי הוא פונקציה עם פרמטרים מוגדרים.

מארח (Host) – רכיב שמתאם בין שרתים. המארח שולח בקשות לשרתים. הוא מקבל תשובות ומעבד אותן.



1.3 תחבורת HTTP על localhost

בתרגיל זה נשתמש בתחבורת HTTP. כל סוכן יפעל על פורט שונה ב-localhost.

1.3.1 הגדרת פורטים

נגדיר פורטים קבועים לכל סוכן:

- League Manager – פורט 8000

- Referee – פורט 8001

- שחקנים – פורטים 8101 עד 8104

כל סוכן מממש שרת HTTP פשוט. השרת מקבל בקשות POST בנתיב /mcp. תוכן הבקשה הוא JSON-RPC 2.0.

1.3.2 דוגמה לכתובת סוכן

כתובת שרת League Manager:

`http://localhost:8000/mcp`

כתובת שרת שחקן ראשון:

`http://localhost:8101/mcp`

1.4 מבנה הודעת JSON-RPC

כל הודעה בפרוטוקול היא אובייקט JSON. להודעה יש מבנה קבוע.

מבנה בסיסי של הודעה

```
{
  "jsonrpc": "2.0",
  "method": "tool_name",
  "params": {
    "param1": "value1",
    "param2": "value2"
  },
  "id": 1
}
```

השדות בהודעה:

- jsonrpc – גרסת הפרוטוקול, תמיד "2.0".

- method – שם הכלי שרוצים להפעיל.

- params – פרמטרים לכלי.

- id – מזהה ייחודי לבקשה.

1.5 מטרת התרגיל

בתרגיל זה נבנה מערכת ליגה לסוכני AI. המערכת תכלול שלושה סוגי סוכנים:

1. **מנהל ליגה (League Manager)** – מנהל את הליגה, כולל רישום שחקנים ושופטים.

2. **שופט (Referee)** – נרשם למנהל הליגה ומנהל משחקים בודדים.

3. **סוכני שחקן (Player Agents)** – משתתפים במשחקים.

תהליך הרישום: לפני תחילת הליגה, גם שופטים וגם שחקנים חייבים להירשם אצל מנהל הליגה. מנהל הליגה שומר רשימה של שופטים זמינים ומקצה אותם למשחקים. המשחק הספציפי בתרגיל הוא "זוגי/אי-זוגי". הפרוטוקול הכללי מאפשר להחליף משחק בעתיד. אפשר יהיה להשתמש באיקס-עיגול, 12 שאלות, או משחקים אחרים.

1.5.1 יעד הלמידה

בסיום התרגיל תוכלו:

- להבין את פרוטוקול MCP.
- לבנות שרת MCP פשוט.
- לתקשר בין סוכנים שונים.
- להריץ ליגה מלאה בסביבה שלכם.

- לוודא תאימות פרוטוקול עם סטודנטים אחרים.

חשוב: כל הסטודנטים ישתמשו באותו פרוטוקול. זה יאפשר לסוכנים שלכם לשחק זה נגד זה בעתיד.

2 פרוטוקול הליגה הכללי

2.1 עקרונות הפרוטוקול

הפרוטוקול מגדיר כללים אחידים. הכללים מאפשרים לסוכנים שונים לתקשר. כל סטודנט יכול לממש סוכן בשפה שירצה. כל עוד הסוכן מכבד את הפרוטוקול – הוא ישתתף בליגה.

2.1.1 הפרדה לשלוש שכבות

המערכת מורכבת משלוש שכבות:

1. **שכבת הליגה** – ניהול טורניר, רישום שחקנים, טבלת דירוג.
 2. **שכבת השיפוט** – ניהול משחק בודד, אימות מהלכים, הכרזת מנצח.
 3. **שכבת חוקי המשחק** – לוגיקת משחק ספציפית (זוגי/אי-זוגי, איקס-עיגול, וכד').
- ההפרדה חשובה. היא מאפשרת להחליף את שכבת המשחק. הפרוטוקול הכללי נשאר קבוע.

2.2 סוגי הסוכנים

2.2.1 מנהל ליגה – League Manager

מנהל הליגה הוא סוכן יחיד. הוא אחראי על:

- רישום שחקנים לליגה.
 - יצירת לוח משחקים (Round-Robin).
 - קבלת תוצאות מהשופטים.
 - חישוב ופרסום טבלת דירוג.
- מנהל הליגה פועל כשרת MCP על פורט 8000.

2.2.2 שופט – Referee

השופט מנהל משחק בודד. **חשוב:** לפני שהשופט יכול לשפוט משחקים, הוא חייב להירשם אצל מנהל הליגה.

השופט אחראי על:

- **רישום למנהל הליגה** – לפני תחילת הליגה.
- הזמנת שני שחקנים למשחק.
- ניהול תורות המשחק.
- אימות חוקיות מהלכים.
- הכרזה על תוצאה ודיווח לליגה.

השופט פועל כשרת MCP על פורט 8001. יכולים להיות מספר שופטים במערכת (פורטים

2.2.3 סוכן שחקן - Player Agent

סוכן השחקן מייצג שחקן בליגה. הוא אחראי על:

- הרשמה לליגה.
 - קבלת הזמנות למשחקים.
 - בחירת מהלכים במשחק.
 - עדכון מצב פנימי לפי תוצאות.
- כל שחקן פועל על פורט נפרד (8101-8104).

2.3 מזהים בפרוטוקול

כל רכיב במערכת מזוהה באופן ייחודי.

טבלה 1: מזהים בפרוטוקול הליגה

שם המזהה	סוג	תיאור
league_id	String	מזהה ליגה ייחודי
round_id	Integer	מספר מחזור בליגה
match_id	String	מזהה משחק בודד
game_type	String	סוג המשחק
player_id	String	מזהה שחקן
referee_id	String	מזהה שופט
conversation_id	String	מזהה שיחה

2.3.1 דוגמאות למזהים

- league_id: "league_2025_even_odd"
- round_id: 1, 2, 3, ...
- match_id: "R1M1" (מחזור 1, משחק 1)
- game_type: "even_odd" או "tic_tac_toe"
- player_id: "P01", "P02", ..., "P20"
- referee_id: "REF01", "REF02", ...

2.4 מבנה הודעה כללי – Envelope

כל הודעה בפרוטוקול חייבת לכלול "מעטפת" (Envelope) עם שדות קבועים. המעטפת מבטיחה עקביות ומאפשרת מעקב אחר ההודעות.

מבנה מעטפת הודעה – Envelope

```
{
  "protocol": "league.v2",
  "message_type": "GAME_INVITATION",
  "sender": "referee:REF01",
  "timestamp": "2025-01-15T10:30:00Z",
  "conversation_id": "conv-r1m1-001",
  "auth_token": "tok_abc123def456...",
  "league_id": "league_2025_even_odd",
  "round_id": 1,
  "match_id": "R1M1"
}
```

2.4.1 שדות חובה במעטפת

טבלה 2: שדות חובה בכל הודעה

שדה	סוג	תיאור
protocol	String	גרסת הפרוטוקול, קבוע "league.v2"
message_type	String	סוג ההודעה (למשל GAME_INVITATION)
sender	String	מזהה השולח בפורמט type:id
timestamp	String	חותמת זמן ISO-8601 באזור זמן UTC
conversation_id	String	מזהה ייחודי שיחה

2.4.2 דרישת אזור זמן – UTC/GMT

חובה: כל חותמות הזמן בפרוטוקול חייבות להיות באזור זמן UTC/GMT. דרישה זו מבטיחה עקביות בין סוכנים הפועלים ממיקומים גיאוגרפיים שונים.

טבלה 3: פורמטים תקינים ולא תקינים לחותמת זמן

פורמט	תקין?	הסבר
2025-01-15T10:30:00Z	✓	סיומת Z מציינת UTC
2025-01-15T10:30:00+00:00	✓	הפרש +00:00 שווה ערך ל-UTC
2025-01-15T10:30:00+02:00	✗	אזור זמן מקומי – אסור
2025-01-15T10:30:00	✗	ללא אזור זמן – אסור

הערה חשובה: סוכן ששולח הודעה עם אזור זמן שאינו UTC יקבל שגיאה E021 (IN-).
(VALID_TIMESTAMP).

2.4.3 שדות אופציונליים

טבלה 4: שדות אופציונליים לפי הקשר

שדה	סוג	תיאור
auth_token	String	טוקן אימות (חובה לאחר רישום)
league_id	String	מזהה הליגה
round_id	Integer	מספר מחזור
match_id	String	מזהה משחק

2.4.4 פורמט שדה sender

שדה sender מזהה את שולח ההודעה:

- league_manager – מנהל הליגה.

- referee:REF01 – שופט עם מזהה REF01.

- player:P01 – שחקן עם מזהה P01.

2.4.5 טוקן אימות - auth_token

לאחר רישום מוצלח, כל סוכן מקבל auth_token. הטוקן חייב להופיע בכל הודעה שנשלחת לאחר הרישום. זה מונע התחזות לסוכנים אחרים.

קבלת טוקן בתגובת רישום

```
{
  "message_type": "LEAGUE_REGISTER_RESPONSE",
  "status": "ACCEPTED",
  "player_id": "P01",
  "auth_token": "tok_p01_abc123def456ghi789..."
}
```

2.5 זרימת הליגה הכללית

2.5.1 שלב 1: רישום שופטים

בשלב הראשון, כל שופט נרשם לליגה. השופט שולח בקשת רישום למנהל הליגה. מנהל הליגה מקצה referee_id ושומר את כתובת השופט.



2.5.2 שלב 2: רישום שחקנים

לאחר רישום השופטים, כל שחקן נרשם לליגה. השחקן שולח בקשת רישום למנהל הליגה. מנהל הליגה מקצה player_id ומאשר.



2.5.3 שלב 3: יצירת לוח משחקים

לאחר שכל השחקנים נרשמו, מנהל הליגה יוצר לוח משחקים. הלוח מבוסס על שיטת Round-Robin. כל שחקן משחק נגד כל שחקן אחר.

2.5.4 שלב 4: הכרזה על מחזור

לפני כל מחזור, מנהל הליגה מפרסם הודעת ROUND_ANNOUNCEMENT. ההודעה מפרטת את כל המשחקים במחזור. מנהל הליגה מקצה שופט לכל משחק מתוך רשימת השופטים הרשומים.

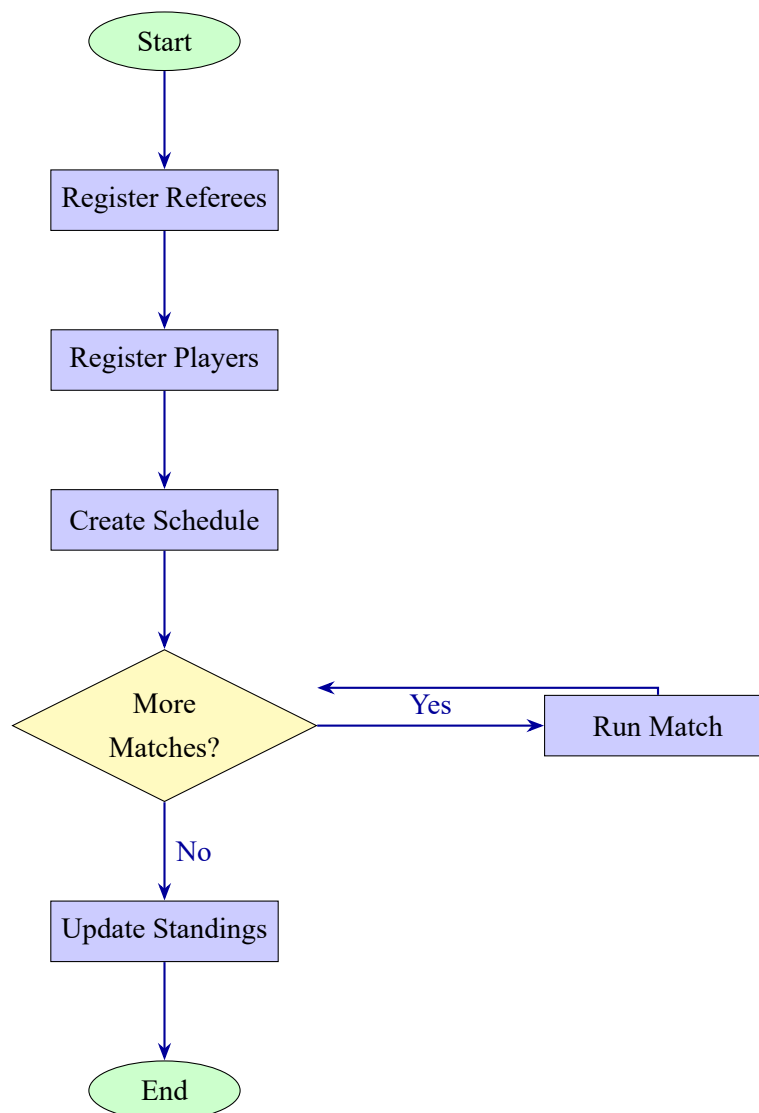
2.5.5 שלב 5: ניהול משחקים

השופט מזמין שחקנים למשחק. הוא מנהל את המשחק לפי חוקי המשחק. בסיום, הוא מדווח תוצאה למנהל הליגה.

2.5.6 שלב 6: עדכון דירוג

לאחר כל מחזור, מנהל הליגה מעדכן את טבלת הדירוג. הוא מפרסם את הטבלה לכל השחקנים.

2.6 דיאגרמת זרימה כללית



2.7 זמני תגובה - Timeouts

לכל סוג הודעה מוגדר זמן תגובה מקסימלי. אם הסוכן לא עונה בזמן, הפעולה נחשבת לכישלון.

טבלה 5: זמני תגובה לפי סוג הודעה

הערות	Timeout	סוג הודעה
רישום שופט לליגה	10 sec	REFEREE_REGISTER
רישום שחקן לליגה	10 sec	LEAGUE_REGISTER
אישור הגעה למשחק	5 sec	GAME_JOIN_ACK
בחירת זוגי/אי-זוגי	30 sec	CHOOSE_PARITY
קבלת תוצאת משחק	5 sec	GAME_OVER
דיווח תוצאה לליגה	10 sec	MATCH_RESULT_REPORT
שאלת מידע	10 sec	LEAGUE_QUERY
ברירת מחדל	10 sec	כל השאר

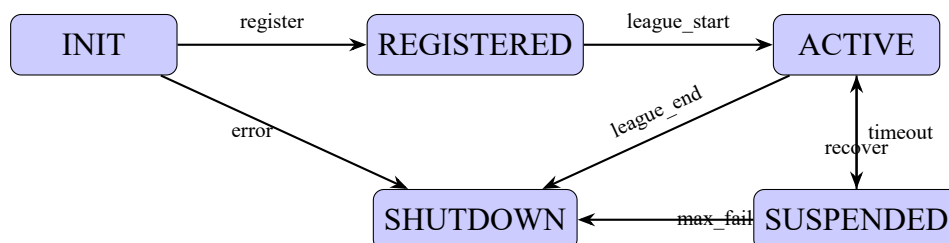
2.8 מחזור חיים של סוכן – Agent Lifecycle

כל סוכן (שחקן, שופט) עובר מצבים מוגדרים במהלך הליגה.

2.8.1 מצבי הסוכן

- INIT – הסוכן הופעל אך טרם נרשם.
- REGISTERED – הסוכן נרשם בהצלחה וקיבל auth_token.
- ACTIVE – הסוכן פעיל ומשתתף במשחקים.
- SUSPENDED – הסוכן מושעה זמנית (לא מגיב).
- SHUTDOWN – הסוכן סיים פעילות.

2.8.2 דיאגרמת מעברי מצבים



2.9 טיפול בשגיאות

הפרוטוקול מגדיר שני סוגי הודעות שגיאה:

2.9.1 שגיאת ליגה – LEAGUE_ERROR

מנהל הליגה שולח הודעה זו כאשר מתרחשת שגיאה ברמת הליגה.

דוגמת שגיאת ליגה

```
{
  "protocol": "league.v2",
  "message_type": "LEAGUE_ERROR",
  "sender": "league_manager",
  "timestamp": "2025-01-15T10:35:00Z",
  "error_code": "E005",
  "error_name": "PLAYER_NOT_REGISTERED",
  "error_description": "Player_ID_not_found_in_registry",
  "context": {
    "player_id": "P99"
  },
  "retryable": false
}
```

2.9.2 שגיאת משחק – GAME_ERROR

השופט שולח הודעה זו כאשר מתרחשת שגיאה במשחק.

דוגמת שגיאת משחק

```
{
  "protocol": "league.v2",
  "message_type": "GAME_ERROR",
  "sender": "referee:REF01",
  "timestamp": "2025-01-15T10:31:00Z",
  "match_id": "R1M1",
  "player_id": "P01",
  "error_code": "E001",
  "error_name": "TIMEOUT_ERROR",
  "error_description": "Response_not_received_within_30_seconds",
  "game_state": "COLLECTING_CHOICES",
  "retryable": true,
  "retry_count": 1,
  "max_retries": 3
}
```

2.9.3 קודי שגיאה נפוצים

טבלה 6: קודי שגיאה עיקריים

קוד	שם	תיאור
E001	TIMEOUT_ERROR	תגובה לא התקבלה בזמן
E003	MISSING_REQUIRED_FIELD	שדה חובה חסר
E004	INVALID_PARITY_CHOICE	בחירה לא חוקית
E005	PLAYER_NOT_REGISTERED	שחקן לא רשום
E009	CONNECTION_ERROR	כשל בחיבור
E011	AUTH_TOKEN_MISSING	טוקן אימות חסר
E012	AUTH_TOKEN_INVALID	טוקן לא תקין

2.9.4 מדיניות ניסיון חוזר – Retry Policy

שגיאות מסוימות ניתנות לניסיון חוזר:

- מקסימום ניסיונות: 3
- השהייה בין ניסיונות: 2 שניות
- שגיאות הניתנות לניסיון: E001 (tuomit), E009 (noitcennoc)
- לאחר מיצוי הניסיונות – הפסד טכני (TECHNICAL_LOSS).

2.10 תאימות גרסאות

2.10.1 הכרזת גרסה

בעת רישום, כל סוכן מכריז על גרסת הפרוטוקול שהוא תומך בה. מנהל הליגה בודק תאימות לפני אישור הרישום.

הכרזת גרסה בבקשת רישום

```
{
  "message_type": "LEAGUE_REGISTER_REQUEST",
  "player_meta": {
    "display_name": "Agent_Alpha",
    "version": "1.0.0",
    "protocol_version": "2.1.0",
    "game_types": ["even_odd"]
  }
}
```

2.10.2 מדיניות תאימות

- גרסה נוכחית: 2.1.0
- גרסה מינימלית נתמכת: 2.0.0
- סוכנים עם גרסה ישנה יותר יקבלו שגיאת (E018- PROTOCOL_VERSION_MISMATCH).

2.11 עקרונות חשובים

2.11.1 מקור אמת יחיד

השופט הוא מקור האמת למצב המשחק. שחקנים לא שומרים מצב משלהם. הם מסתמכים על המידע שהשופט שולח.

2.11.2 תקשורת דרך Orchestrator

שחקנים לא מדברים ישירות זה עם זה. כל התקשורת עוברת דרך השופט או מנהל הליגה. זה מבטיח שהפרוטוקול נשמר.

2.11.3 טיפול בכשלים

אם שחקן לא מגיב:

1. השופט שולח הודעת GAME_ERROR עם retryable=true.
2. השחקן מקבל עד 3 ניסיונות.
3. לאחר מיצוי הניסיונות – הפסד טכני (TECHNICAL_LOSS).

3 משחק זוגי/אי-זוגי

3.1 תיאור המשחק

משחק זוגי/אי-זוגי הוא משחק פשוט. המשחק מתאים להדגמת פרוטוקול הליגה.

3.1.1 חוקי המשחק

1. שני שחקנים משתתפים במשחק.
2. כל שחקן בוחר "זוגי" (even) או "אי-זוגי" (odd).
3. הבחירות נעשות במקביל, בלי לדעת את בחירת היריב.
4. השופט מגריל מספר בין 1 ל-10.
5. אם המספר זוגי – מי שבחר "זוגי" מנצח.
6. אם המספר אי-זוגי – מי שבחר "אי-זוגי" מנצח.
7. אם שניהם בחרו אותו דבר ופספסו – תיקו.

3.1.2 דוגמה למשחק

נניח משחק בין שחקן A לשחקן B:

טבלה 7: דוגמה למשחק זוגי/אי-זוגי

תוצאה	מספר	בחירת B	בחירת A
A מנצח	8 (even)	odd	even
B מנצח	7 (odd)	odd	even
תיקו	4 (even)	odd	odd

3.2 זרימת משחק בודד

3.2.1 שלב 1: הזמנה למשחק

השופט שולח הזמנה לשני השחקנים. ההזמנה כוללת:

- מזהה המשחק (match_id).
- מזהה המחזור (round_id).
- סוג המשחק (game_type).

3.2.2 שלב 2: אישור הגעה

כל שחקן מאשר קבלת ההזמנה. האישור כולל חותמת זמן.

3.2.3 שלב 3: איסוף בחירות

השופט פונה לכל שחקן בנפרד. הוא מבקש בחירה: "זוגי" או "אי-זוגי". השחקן מחזיר את בחירתו.

חשוב: השחקנים לא רואים את בחירת היריב.

3.2.4 שלב 4: הגרלת מספר

לאחר קבלת שתי הבחירות, השופט מגריל מספר. המספר הוא בין 1 ל-10. ההגרלה חייבת להיות אקראית.

3.2.5 שלב 5: קביעת מנצח

השופט בודק:

- אם המספר זוגי ושחקן בחר "זוגי" – הוא מנצח.
- אם המספר אי-זוגי ושחקן בחר "אי-זוגי" – הוא מנצח.
- אם שניהם ניחשו נכון/לא נכון – תיקו.

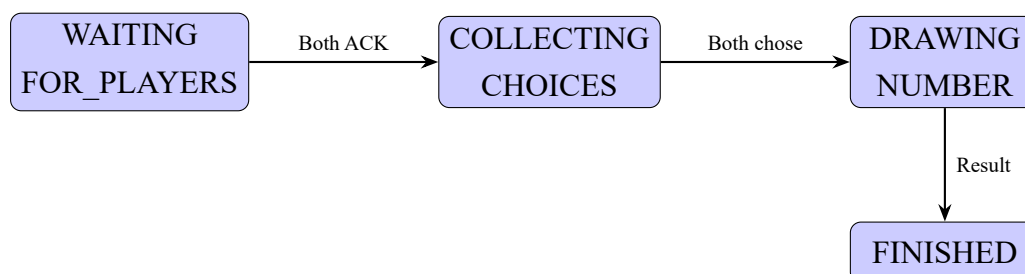
3.2.6 שלב 6: דיווח תוצאה

השופט שולח:

1. הודעת GAME_OVER לשני השחקנים.
2. הודעת MATCH_RESULT_REPORT למנהל הליגה.

3.3 מצבי המשחק

המשחק עובר בין מצבים מוגדרים:



3.3.1 מצב WAITING_FOR_PLAYERS

המשחק מתחיל במצב זה. השופט מחכה שהשחקנים יאשרו הגעה. המעבר: כששני השחקנים שלחו GAME_JOIN_ACK.

3.3.2 מצב COLLECTING_CHOICES

השופט אוסף בחירות מהשחקנים. הוא קורא ל-choose_parity של כל שחקן. המעבר: כששתי הבחירות התקבלו.

3.3.3 מצב DRAWING_NUMBER

השופט מגריל מספר וקובע מנצח. המעבר: אוטומטי לאחר החישוב.

3.3.4 מצב FINISHED

המשחק הסתיים. התוצאה דווחה.

3.4 שיטת הניקוד

3.4.1 ניקוד למשחק

טבלה 8: טבלת ניקוד

תוצאה	נקודות מנצח	נקודות מפסיד
ניצחון	3	0
תיקו	1	1
הפסד	0	0

3.4.2 דירוג בליגה

הדירוג נקבע לפי:

1. סך הנקודות (יורד).
2. מספר הניצחונות (יורד).
3. הפרש תיקו (יורד).

3.5 ליגת Round-Robin

בליגה עם 4 שחקנים, כל שחקן משחק נגד כולם.

3.5.1 מספר משחקים

עבור n שחקנים:

- מספר משחקים בליגה: $\frac{n(n-1)}{2}$

- עבור 4 שחקנים: $\frac{4 \times 3}{2} = 6$ משחקים

3.5.2 לוח משחקים לדוגמה

טבלה 9: לוח משחקים ל-4 שחקנים

משחק	שחקן A	שחקן B
R1M1	P01	P02
R1M2	P03	P04
R2M1	P01	P03
R2M2	P02	P04
R3M1	P01	P04
R3M2	P02	P03

3.6 אסטרטגיות לשחקנים

3.6.1 אסטרטגיה אקראית

הגישה הפשוטה ביותר. השחקן בוחר באקראי "זוגי" או "אי-זוגי". הסיכוי לנצח הוא 50%.

אסטרטגיה אקראית

```
import random

def choose_parity_random():
    return random.choice(["even", "odd"])
```

3.6.2 אסטרטגיה מבוססת היסטוריה

השחקן זוכר תוצאות קודמות. הוא מנסה לזהות דפוסים בהגרלות. **הערה:** מכיוון שההגרלה אקראית, אסטרטגיה זו לא תשפר תוצאות לטווח ארוך.

3.6.3 אסטרטגיה מונחית LLM

השחקן יכול להשתמש במודל שפה. הוא בונה prompt ושואל את המודל.

דוגמה ל-prompt

```
prompt = """
You are playing Even/Odd game.
Choose "even" or "odd".
Previous results: even won 3 times, odd won 2 times.
Your choice (one word only):
"""
```

הערה: השימוש ב-LLM מעניין אך לא ישפר ביצועים סטטיסטית. המשחק הוא משחק מזל.

3.7 מודול חוקי המשחק

מודול החוקים הוא רכיב נפרד בשופט. הוא מגדיר את הלוגיקה הספציפית למשחק.

3.7.1 ממשק המודול

המודול מספק פונקציות:

- `init_game_state()` – אתחול מצב משחק.
- `validate_choice(choice)` – בדיקת חוקיות בחירה.
- `draw_number()` – הגרלת מספר.
- `determine_winner(choices, number)` – קביעת מנצח.

3.7.2 יתרון ההפרדה

בעתיד, אפשר להחליף את המודול. במקום זוגי/אי-זוגי, אפשר:

- איקס-עיגול (Tic-Tac-Toe).
- 21 שאלות.
- משחק זיכרון.

הפרוטוקול הכללי נשאר זהה. רק מודול החוקים משתנה.

3.8 הרחבה למשחקים נוספים

הפרוטוקול מתוכנן להיות כללי ולא ספציפי למשחק זוגי/אי-זוגי. סעיף זה מתאר את השכבה הגנרית המאפשרת הוספת משחקים נוספים.

3.8.1 הפשטת מהלך – `GAME_MOVE`

הודעות `CHOOSE_PARITY_CALL` ו-`CHOOSE_PARITY_RESPONSE` הן מקרה פרטי של הפשטה כללית יותר:

טבלה 10: הקבלה בין הודעות ספציפיות לגנריות

הודעה ספציפית	הודעה גנרית
<code>CHOOSE_PARITY_CALL</code>	<code>GAME_MOVE_CALL</code>
<code>CHOOSE_PARITY_RESPONSE</code>	<code>GAME_MOVE_RESPONSE</code>

GAME_MOVE_CALL - בקשת מהלך גנרית

```
{
  "protocol": "league.v2",
  "message_type": "GAME_MOVE_CALL",
  "sender": "referee:REF01",
  "timestamp": "2025-01-15T10:30:15Z",
  "match_id": "R1M1",
  "player_id": "P01",
  "game_type": "even_odd",
  "move_request": {
    "move_type": "choose_parity",
    "valid_options": ["even", "odd"],
    "context": {}
  },
  "deadline": "2025-01-15T10:30:45Z"
}
```

GAME_MOVE_RESPONSE - תגובת מהלך גנרית

```
{
  "protocol": "league.v2",
  "message_type": "GAME_MOVE_RESPONSE",
  "sender": "player:P01",
  "timestamp": "2025-01-15T10:30:20Z",
  "match_id": "R1M1",
  "player_id": "P01",
  "game_type": "even_odd",
  "move_data": {
    "move_type": "choose_parity",
    "choice": "even"
  }
}
```

3.8.3 רישום סוגי משחקים – Game Registry

מנהל הליגה מחזיק רישום של סוגי משחקים נתמכים:

רישום משחקים

```
{
  "game_registry": {
    "even_odd": {
      "display_name": "Even/Odd",
      "move_types": ["choose_parity"],
      "valid_choices": {
        "choose_parity": ["even", "odd"]
      },
      "min_players": 2,
      "max_players": 2
    },
    "tic_tac_toe": {
      "display_name": "Tic-Tac-Toe",
      "move_types": ["place_mark"],
      "valid_choices": {
        "place_mark": ["0-8"]
      },
      "min_players": 2,
      "max_players": 2
    }
  }
}
```

3.8.4 יתרונות ההפשטה

1. **הוספת משחקים חדשים** – ללא שינוי בפרוטוקול הבסיסי.
 2. **גילוי יכולות** – שחקן יכול לשאול אילו משחקים נתמכים.
 3. **וולידציה אחידה** – השופט מוודא שהמהלך חוקי לפי ה-schema.
 4. **תאימות קדימה** – סוכנים ישנים יכולים להמשיך לעבוד עם הודעות ספציפיות.
- הערה:** בתרגיל זה משתמשים בהודעות הספציפיות (CHOOSE_PARITY_*). ההפשטה הגנרית מוצגת לצורך הבנת הארכיטקטורה.

4 מבני הודעות JSON

פרק זה מגדיר את כל הודעות הפרוטוקול. **חשוב מאוד:** כל הסטודנטים חייבים להשתמש במבנים אלה בדיוק. זה יאפשר לסוכנים שלכם לתקשר זה עם זה.

4.1 הודעות רישום שופט לליגה

4.1.1 REFEREE_REGISTER_REQUEST – בקשת רישום שופט

- מאת (From): referee (שופט)
 - אל (To): league_manager (מנהל הליגה)
 - תגובה צפויה: REFEREE_REGISTER_RESPONSE
- שופט שולח בקשה זו למנהל הליגה לפני תחילת הליגה.

בקשת רישום שופט לליגה

```
{
  "message_type": "REFEREE_REGISTER_REQUEST",
  "referee_meta": {
    "display_name": "Referee_Alpha",
    "version": "1.0.0",
    "game_types": ["even_odd"],
    "contact_endpoint": "http://localhost:8001/mcp",
    "max_concurrent_matches": 2
  }
}
```

שדות חובה:

- display_name – שם תצוגה של השופט.
- version – גרסת השופט.
- game_types – רשימת סוגי משחקים שהשופט יודע לשפוט.
- contact_endpoint – כתובת השרת של השופט.
- max_concurrent_matches – מספר משחקים מקסימלי שהשופט יכול לנהל במקביל.

4.1.2 תגובת רישום שופט – REFEREE_REGISTER_RESPONSE

- מאת (From): league_manager (מנהל הליגה)
 - אל (To): referee (השופט ששלח את הבקשה)
 - תגובה צפויה: אין (הודעת תגובה)
- מנהל הליגה מחזיר תגובה זו לשופט.

תגובת רישום שופט לליגה

```
{  
  "message_type": "REFEREE_REGISTER_RESPONSE",  
  "status": "ACCEPTED",  
  "referee_id": "REF01",  
  "reason": null  
}
```

שדות:

- status – "ACCEPTED" או "REJECTED".
- referee_id – מזהה שהוקצה לשופט (רק אם התקבל).
- reason – סיבת דחייה (רק אם נדחה).

4.2 הודעות רישום שחקן לליגה

4.2.1 בקשת רישום שחקן – LEAGUE_REGISTER_REQUEST

- מאת (From): player (שחקן)
 - אל (To): league_manager (מנהל הליגה)
 - תגובה צפויה: LEAGUE_REGISTER_RESPONSE
- שחקן שולח בקשה זו למנהל הליגה.

בקשת רישום לליגה

```
{
  "message_type": "LEAGUE_REGISTER_REQUEST",
  "player_meta": {
    "display_name": "Agent_Alpha",
    "version": "1.0.0",
    "game_types": ["even_odd"],
    "contact_endpoint": "http://localhost:8101/mcp"
  }
}
```

שדות חובה:

- display_name – שם תצוגה של השחקן.
- version – גרסת הסוכן.
- game_types – רשימת משחקים נתמכים.
- contact_endpoint – כתובת השרת של השחקן.

4.2.2 תגובת רישום – LEAGUE_REGISTER_RESPONSE

- מאת (From): league_manager (מנהל הליגה)

- אל (To): player (השחקן ששלח את הבקשה)

- תגובה צפויה: אין (הודעת תגובה)

מנהל הליגה מחזיר תגובה זו.

תגובת רישום לליגה

```
{
  "message_type": "LEAGUE_REGISTER_RESPONSE",
  "status": "ACCEPTED",
  "player_id": "P01",
  "reason": null
}
```

שדות:

- status – "ACCEPTED" או "REJECTED".

- player_id – מזהה שהוקצה לשחקן (רק אם התקבל).

- reason – סיבת דחייה (רק אם נדחה).

4.3 הודעות מחזור

4.3.1 הכרזת מחזור – ROUND_ANNOUNCEMENT

- מאת (From): league_manager (מנהל הליגה)
 - אל (To): players (כל השחקנים הרשומים)
 - תגובה צפויה: אין (הודעת שידור)
- מנהל הליגה שולח לפני כל מחזור.

הכרזת מחזור

```
{
  "message_type": "ROUND_ANNOUNCEMENT",
  "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"
    }
  ]
}
```

4.4 הודעות משחק

4.4.1 הזמנה למשחק – GAME_INVITATION

- מאת (From): referee (השופט המנהל את המשחק)
 - אל (To): player (כל אחד משני השחקנים במשחק)
 - תגובה צפויה: GAME_JOIN_ACK
- השופט שולח לכל שחקן.

הזמנה למשחק

```
{
  "message_type": "GAME_INVITATION",
  "league_id": "league_2025_even_odd",
  "round_id": 1,
  "match_id": "R1M1",
  "game_type": "even_odd",
  "role_in_match": "PLAYER_A",
  "opponent_id": "P02",
  "conversation_id": "conv-r1m1-001"
}
```

4.4.2 אישור הגעה – GAME_JOIN_ACK

- מאת (From): player (השחקן שקיבל הזמנה)
 - אל (To): referee (השופט ששלח את ההזמנה)
 - תגובה צפויה: CHOOSE_PARITY_CALL (לאחר שכל השחקנים מאשרים)
- השחקן מאשר קבלת ההזמנה.

אישור הגעה למשחק

```
{
  "message_type": "GAME_JOIN_ACK",
  "match_id": "R1M1",
  "player_id": "P01",
  "arrival_timestamp": "2025-01-15T10:30:00Z",
  "accept": true
}
```

4.5 הודעות בחירה במשחק זוגי/אי-זוגי

4.5.1 CHOOSE_PARITY_CALL – בקשת בחירה

- מאת (From): referee (השופט)
 - אל (To): player (כל אחד מהשחקנים במשחק)
 - תגובה צפויה: CHOOSE_PARITY_RESPONSE
- השופט מבקש מהשחקן לבחור.

בקשת בחירה

```
{
  "message_type": "CHOOSE_PARITY_CALL",
  "match_id": "R1M1",
  "player_id": "P01",
  "game_type": "even_odd",
  "context": {
    "opponent_id": "P02",
    "round_id": 1,
    "your_standings": {
      "wins": 2,
      "losses": 1,
      "draws": 0
    }
  },
  "deadline": "2025-01-15T10:30:30Z"
}
```

4.5.2 CHOOSE_PARITY_RESPONSE – תגובת בחירה

- מאת (From): player (השחקן)
 - אל (To): referee (השופט ששלח את הבקשה)
 - תגובה צפויה: GAME_OVER (לאחר שכל השחקנים ענו)
- השחקן מחזיר את בחירתו.

תגובת בחירה

```
{
  "message_type": "CHOOSE_PARITY_RESPONSE",
  "match_id": "R1M1",
  "player_id": "P01",
  "parity_choice": "even"
}
```

חשוב: הערך של parity_choice חייב להיות "even" או "odd" בדיוק.

4.6 הודעות תוצאה

4.6.1 סיום משחק – GAME_OVER

- מאת (From): referee (השופט)
 - אל (To): players (שני השחקנים במשחק)
 - תגובה צפויה: אין (הודעת עדכון)
- השופט שולח לשני השחקנים.

הודעת סיום משחק

```
{
  "message_type": "GAME_OVER",
  "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)"
  }
}
```

ערכים אפשריים ל-status:

- "WIN" – יש מנצח.
- "DRAW" – תיקו.
- "TECHNICAL_LOSS" – הפסד טכני (זמן תגובה, וכד').

4.6.2 MATCH_RESULT_REPORT – דיווח תוצאה לליגה

- מאת (From): referee (השופט שניהל את המשחק)
- אל (To): league_manager (מנהל הליגה)
- תגובה צפויה: LEAGUE_STANDINGS_UPDATE (מנהל הליגה ישדר לכל השחקנים)
השופט שולח למנהל הליגה.

דיווח תוצאה לליגה

```
{
  "message_type": "MATCH_RESULT_REPORT",
  "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"
      }
    }
  }
}
```

4.7 הודעות דירוג

4.7.1 LEAGUE_STANDINGS_UPDATE - עדכון דירוג

- מאת (From): league_manager (מנהל הליגה)
 - אל (To): players (כל השחקנים הרשומים)
 - תגובה צפויה: אין (הודעת שידור)
- מנהל הליגה שולח לכל השחקנים.

עדכון טבלת דירוג

```
{
  "message_type": "LEAGUE_STANDINGS_UPDATE",
  "league_id": "league_2025_even_odd",
  "round_id": 1,
  "standings": [
    {
      "rank": 1,
      "player_id": "P01",
      "display_name": "Agent_αAlpha",
      "played": 2,
      "wins": 2,
      "draws": 0,
      "losses": 0,
      "points": 6
    },
    {
      "rank": 2,
      "player_id": "P03",
      "display_name": "Agent_γGamma",
      "played": 2,
      "wins": 1,
      "draws": 1,
      "losses": 0,
      "points": 4
    }
  ]
}
```

4.8 הודעות סיום מחזור וליגה

4.8.1 סיום מחזור - ROUND_COMPLETED

- מאת (From): league_manager (מנהל הליגה)

- אל (To): players (כל השחקנים הרשומים)

- תגובה צפויה: אין (הודעת שידור)

מנהל הליגה שולח הודעה זו לכל המשתתפים בסיום מחזור.

הודעת סיום מחזור

```
{
  "protocol": "league.v2",
  "message_type": "ROUND_COMPLETED",
  "sender": "league_manager",
  "timestamp": "2025-01-15T12:00:00Z",
  "conversation_id": "conv-round1-complete",
  "league_id": "league_2025_even_odd",
  "round_id": 1,
  "matches_completed": 2,
  "next_round_id": 2,
  "summary": {
    "total_matches": 2,
    "wins": 1,
    "draws": 1,
    "technical_losses": 0
  }
}
```

שדות:

- round_id - המחזור שהסתיים.

- matches_completed - מספר משחקים שהושלמו.

- next_round_id - המחזור הבא, או null אם זה המחזור האחרון.

- summary - סיכום סטטיסטי של המחזור.

4.8.2 סיום ליגה – LEAGUE_COMPLETED

- מאת (From): league_manager (מנהל הליגה)
 - אל (To): all_agents (כל הסוכנים – שחקנים ושופטים)
 - תגובה צפויה: אין (הודעת שידור סופית)
- מנהל הליגה שולח הודעה זו לכל הסוכנים בסיום הליגה.

הודעת סיום ליגה

```
{
  "protocol": "league.v2",
  "message_type": "LEAGUE_COMPLETED",
  "sender": "league_manager",
  "timestamp": "2025-01-20T18:00:00Z",
  "conversation_id": "conv-league-complete",
  "league_id": "league_2025_even_odd",
  "total_rounds": 3,
  "total_matches": 6,
  "champion": {
    "player_id": "P01",
    "display_name": "Agent Alpha",
    "points": 9
  },
  "final_standings": [
    {"rank": 1, "player_id": "P01", "points": 9},
    {"rank": 2, "player_id": "P03", "points": 5},
    {"rank": 3, "player_id": "P02", "points": 3},
    {"rank": 4, "player_id": "P04", "points": 1}
  ]
}
```

שדות:

- champion – פרטי האלוף.
- final_standings – טבלת הדירוג הסופית.
- total_matches, total_rounds – סטטיסטיקות הליגה.

4.9 הודעות שאילתא

4.9.1 שאלתת ליגה – LEAGUE_QUERY

- מאת (From): player או referee (שחקן או שופט)
 - אל (To): league_manager (מנהל הליגה)
 - תגובה צפויה: LEAGUE_QUERY_RESPONSE
- שחקן או שופט שולח שאלתת למנהל הליגה לקבלת מידע.

שאלתת המשחק הבא

```
{
  "protocol": "league.v2",
  "message_type": "LEAGUE_QUERY",
  "sender": "player:P01",
  "timestamp": "2025-01-15T14:00:00Z",
  "conversation_id": "conv-query-001",
  "auth_token": "tok_p01_abc123...",
  "league_id": "league_2025_even_odd",
  "query_type": "GET_NEXT_MATCH",
  "query_params": {
    "player_id": "P01"
  }
}
```

סוגי שאלתות (query_type):

- GET_STANDINGS – קבלת טבלת דירוג.
- GET_SCHEDULE – קבלת לוח משחקים.
- GET_NEXT_MATCH – קבלת פרטי המשחק הבא.
- GET_PLAYER_STATS – קבלת סטטיסטיקות שחקן.

4.9.2 תגובת שאילתה – LEAGUE_QUERY_RESPONSE

- מאת (From): league_manager (מנהל הליגה)
 - אל (To): player או referee (השולח המקורי)
 - תגובה צפויה: אין (הודעת תגובה)
- מנהל הליגה מחזיר תגובה לשאילתה.

תגובת המשחק הבא

```
{
  "protocol": "league.v2",
  "message_type": "LEAGUE_QUERY_RESPONSE",
  "sender": "league_manager",
  "timestamp": "2025-01-15T14:00:01Z",
  "conversation_id": "conv-query-001",
  "query_type": "GET_NEXT_MATCH",
  "success": true,
  "data": {
    "next_match": {
      "match_id": "R2M1",
      "round_id": 2,
      "opponent_id": "P03",
      "referee_endpoint": "http://localhost:8001/mcp"
    }
  }
}
```

שדות:

- success – האם השאילתה הצליחה.
- data – תוצאת השאילתה (מבנה משתנה לפי query_type).
- error – פרטי שגיאה אם success=false.

4.10 הודעות שגיאה

4.10.1 LEAGUE_ERROR – שגיאה ברמת הליגה

- מאת (From): league_manager (מנהל הליגה)
- אל (To): agent (הסוכן שגרם לשגיאה)
- תגובה צפויה: אין (הודעת שגיאה)

כאשר מתרחשת שגיאה בפעולות ליגה, מנהל הליגה שולח הודעת LEAGUE_ERROR:

שגיאת ליגה – LEAGUE_ERROR

```
{
  "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",
  "original_message_type": "LEAGUE_QUERY",
  "context": {
    "provided_token": "tok-invalid-xxx",
    "expected_format": "tok-{agent_id}-{hash}"
  }
}
```

שדות:

- error_code – קוד השגיאה מטבלת קודי השגיאה.
- error_description – שם השגיאה.
- original_message_type – סוג ההודעה שגרמה לשגיאה.
- context – מידע נוסף לצורך ניפוי.

4.10.2 GAME_ERROR – שגיאה ברמת המשחק

- מאת (From): referee (השופט המנהל את המשחק)
- אל (To): player (השחקן שגרם לשגיאה או המושפע ממנה)
- תגובה צפויה: אין (הודעת שגיאה)

כאשר מתרחשת שגיאה במהלך משחק, השופט שולח הודעת GAME_ERROR:

שגיאת משחק – GAME_ERROR

```
{
  "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_info": {
    "retry_count": 1,
    "max_retries": 3,
    "next_retry_at": "2025-01-15T10:16:02Z"
  },
  "consequence": "Technical_loss_if_max_retries_exceeded"
}
```

שדות:

- match_id – מזהה המשחק בו אירעה השגיאה.
- affected_player – השחקן המושפע.
- action_required – הפעולה שנכשלה.
- retry_info – מידע על ניסיונות חוזרים (אם רלוונטי).
- consequence – התוצאה אם השגיאה לא תיפתר.

4.11 טבלת סיכום הודעות

4.12 כללים חשובים

טבלה 11: סיכום כל 18 סוגי ההודעות בפרוטוקול v2.1

סוג הודעה	שולח	מקבל	מטרה
REFEREE_REGISTER_REQUEST	שופט	ליגה	רישום שופט
REFEREE_REGISTER_RESPONSE	ליגה	שופט	רישום אישור שופט
LEAGUE_REGISTER_REQUEST	שחקן	ליגה	רישום שחקן
LEAGUE_REGISTER_RESPONSE	ליגה	שחקן	רישום אישור שחקן
ROUND_ANNOUNCEMENT	ליגה	שחקנים	פרסום מחזור
ROUND_COMPLETED	ליגה	שחקנים	סיום מחזור
LEAGUE_COMPLETED	ליגה	כולם	סיום ליגה
GAME_INVITATION	שופט	שחקן	הזמנה למשחק
GAME_JOIN_ACK	שחקן	שופט	אישור הגעה
CHOOSE_PARITY_CALL	שופט	שחקן	בקשת בחירה
CHOOSE_PARITY_RESPONSE	שחקן	שופט	תגובת בחירה
GAME_OVER	שופט	שחקנים	סיום משחק
MATCH_RESULT_REPORT	שופט	ליגה	דיווח תוצאה
LEAGUE_STANDINGS_UPDATE	ליגה	שחקנים	עדכון דירוג
LEAGUE_ERROR	ליגה	סוכן	שגיאת ליגה
GAME_ERROR	שופט	שחקן	שגיאת משחק
LEAGUE_QUERY	שחקן/שופט	ליגה	שאלת מידע
LEAGUE_QUERY_RESPONSE	ליגה	שחקן/שופט	תגובת שאלתה

4.12.1 שדות חובה

כל הודעה חייבת לכלול:

- message_type - תמיד.
- match_id - בהודעות משחק.
- player_id - בהודעות שחקן.

4.12.2 ערכים מותרים

- parity_choice: רק "even" או "odd" (אותיות קטנות).
- status: רק "WIN", "DRAW", או "TECHNICAL_LOSS".
- accept: רק true או false (בוליאני).

4.12.3 פורמט זמן

כל חותמות הזמן בפורמט ISO-8601:

YYYY-MM-DDTHH:MM:SSZ

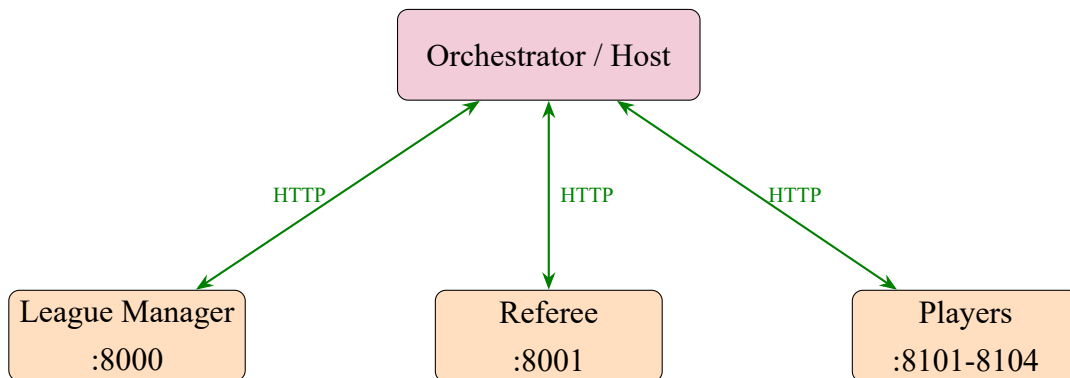
דוגמה: 2025-01-15T10:30:00Z

5 מדריך מימוש

פרק זה מציג כיצד לממש את הסוכנים. הדוגמאות ב-Python עם FastAPI. אפשר להשתמש בכל שפה שתומכת ב-HTTP.

5.1 ארכיטקטורה כללית

5.1.1 תרשים רכיבים



5.1.2 תפקיד ה-Orchestrator

ה-Orchestrator מתאם בין כל הסוכנים. הוא:

- שולח בקשות HTTP לכל שרת.
- מקבל תגובות ומעבד אותן.
- מנהל את זרימת הליגה.

5.2 מימוש שרת MCP פשוט

5.2.1 מבנה בסיסי ב-FastAPI

שרת MCP בסיסי

```
from fastapi import FastAPI
from pydantic import BaseModel
import uvicorn

app = FastAPI()

class MCPRequest(BaseModel):
    jsonrpc: str = "2.0"
    method: str
    params: dict = {}
    id: int = 1

class MCPResponse(BaseModel):
    jsonrpc: str = "2.0"
    result: dict = {}
    id: int = 1

@app.post("/mcp")
async def mcp_endpoint(request: MCPRequest):
    if request.method == "tool_name":
        result = handle_tool(request.params)
        return MCPResponse(result=result, id=request.id)
    return MCPResponse(result={"error": "Unknown method"})

if __name__ == "__main__":
    uvicorn.run(app, host="localhost", port=8101)
```

5.3 מימוש סוכן שחקן

5.3.1 כלים נדרשים

סוכן שחקן חייב לממש את הכלים הבאים:

1. handle_game_invitation – קבלת הזמנה למשחק.
2. choose_parity – בחירת "זוגי" או "אי-זוגי".
3. notify_match_result – קבלת תוצאת משחק.

סוכן שחקן פשוט

```

import random
from fastapi import FastAPI
from pydantic import BaseModel
from datetime import datetime

app = FastAPI()

class MCPRequest(BaseModel):
    jsonrpc: str = "2.0"
    method: str
    params: dict = {}
    id: int = 1

@app.post("/mcp")
async def mcp_endpoint(request: MCPRequest):
    if request.method == "handle_game_invitation":
        return handle_invitation(request.params)
    elif request.method == "choose_parity":
        return handle_choose_parity(request.params)
    elif request.method == "notify_match_result":
        return handle_result(request.params)
    return {"error": "Unknown method"}

def handle_invitation(params):
    # Accept the invitation
    return {
        "message_type": "GAME_JOIN_ACK",
        "match_id": params.get("match_id"),
        "arrival_timestamp": datetime.now().isoformat(),
        "accept": True
    }

def handle_choose_parity(params):
    # Random strategy
    choice = random.choice(["even", "odd"])
    return {
        "message_type": "CHOOSE_PARITY_RESPONSE",
        "match_id": params.get("match_id"),
        "player_id": params.get("player_id"),
        "parity_choice": choice
    }

def handle_result(params):
    # Log result for learning
    print(f"Match result: {params}")
    return {"status": "ok"}

```

5.4 מימוש שופט

5.4.1 כלים נדרשים

השופט חייב לממש:

1. register_to_league – רישום עצמי למנהל הליגה.
2. start_match – התחלת משחק חדש.
3. collect_choices – איסוף בחירות משחקנים.
4. draw_number – הגרלת מספר.
5. finalize_match – סיום המשחק ודיווח.

5.4.2 רישום שופט לליגה

שופט נרשם למנהל הליגה

```
import requests

def register_to_league(league_endpoint, referee_info):
    payload = {
        "jsonrpc": "2.0",
        "method": "register_referee",
        "params": {
            "referee_meta": {
                "display_name": referee_info["name"],
                "version": "1.0.0",
                "game_types": ["even_odd"],
                "contact_endpoint": referee_info["endpoint"],
                "max_concurrent_matches": 2
            }
        },
        "id": 1
    }
    response = requests.post(league_endpoint, json=payload)
    result = response.json()
    return result.get("result", {}).get("referee_id")
```

קביעת מנצח במשחק זוגי/אי-זוגי

```
def determine_winner(choice_a, choice_b, number):
    is_even = (number % 2 == 0)
    parity = "even" if is_even else "odd"

    a_correct = (choice_a == parity)
    b_correct = (choice_b == parity)

    if a_correct and not b_correct:
        return "PLAYER_A"
    elif b_correct and not a_correct:
        return "PLAYER_B"
    else:
        return "DRAW"
```

5.5 מימוש מנהל ליגה**5.5.1 כלים נדרשים**

מנהל הליגה חייב לממש:

1. register_referee – רישום שופט חדש.
2. register_player – רישום שחקן חדש.
3. create_schedule – יצירת לוח משחקים.
4. report_match_result – קבלת דיווח תוצאה.
5. get_standings – החזרת טבלת דירוג.

רישום שופט במנהל הליגה

```

class LeagueManager:
    def __init__(self):
        self.referees = {} # referee_id -> referee_info
        self.players = {} # player_id -> player_info
        self.next_referee_id = 1

    def register_referee(self, params):
        referee_meta = params.get("referee_meta", {})
        referee_id = f"REF{self.next_referee_id:02d}"
        self.next_referee_id += 1

        self.referees[referee_id] = {
            "referee_id": referee_id,
            "display_name": referee_meta.get("display_name"),
            "endpoint": referee_meta.get("contact_endpoint"),
            "game_types": referee_meta.get("game_types", []),
            "max_concurrent": referee_meta.get("max_concurrent_matches",
                                                1)
        }

    def return {
        "message_type": "REFEREE_REGISTER_RESPONSE",
        "status": "ACCEPTED",
        "referee_id": referee_id,
        "reason": None
    }

```


אלגוריתם Round-Robin

```
from itertools import combinations

def create_schedule(players):
    matches = []
    round_num = 1
    match_num = 1

    for p1, p2 in combinations(players, 2):
        matches.append({
            "match_id": f"R{round_num}M{match_num}",
            "player_A_id": p1,
            "player_B_id": p2
        })
        match_num += 1

    return matches
```

5.6 שליחת בקשות HTTP

5.6.1 קריאה לכלי MCP

שליחת בקשה לשרת MCP

```
import requests

def call_mcp_tool(endpoint, method, params):
    payload = {
        "jsonrpc": "2.0",
        "method": method,
        "params": params,
        "id": 1
    }
    response = requests.post(endpoint, json=payload)
    return response.json()

# Example: Call player's choose_parity
result = call_mcp_tool(
    "http://localhost:8101/mcp",
    "choose_parity",
    {"match_id": "R1M1", "player_id": "P01"}
)
```

5.7 ניהול מצב

5.7.1 מצב שחקן

השחקן יכול לשמור מידע פנימי:

- היסטוריית משחקים.
- סטטיסטיקות אישיות.
- מידע על יריבים.

מחלקת מצב שחקן

```
class PlayerState:
    def __init__(self, player_id):
        self.player_id = player_id
        self.wins = 0
        self.losses = 0
        self.draws = 0
        self.history = []

    def update(self, result):
        self.history.append(result)
        if result["winner"] == self.player_id:
            self.wins += 1
        elif result["winner"] == "DRAW":
            self.draws += 1
        else:
            self.losses += 1
```

5.8 טיפול בשגיאות

5.8.1 זמן תגובה

בקשה עם timeout

```
import requests

def call_with_timeout(endpoint, method, params, timeout=30):
    try:
        response = requests.post(
            endpoint,
            json={"jsonrpc": "2.0", "method": method,
                  "params": params, "id": 1},
            timeout=timeout
        )
        return response.json()
    except requests.Timeout:
        return {"error": "TIMEOUT"}
    except requests.RequestException as e:
        return {"error": str(e)}
```

5.8.2 תגובה לשגיאות

אם שחקן לא עונה:

1. השופט ממתין עד ל-timeout.
2. אם אין תגובה – הפסד טכני.
3. השופט מדווך למנהל הליגה.

5.9 דפוסי חוסן (Resilience Patterns)

מערכת מבוזרת חייבת להתמודד עם כשלים זמניים. הפרוטוקול מגדיר מדיניות ניסיונות חוזרים:

- מקסימום 3 ניסיונות חוזרים.
- השהייה של 2 שניות בין ניסיונות.
- Exponential backoff מומלץ למערכות בעומס.

לוגיקת ניסיונות חוזרים

```
import time
import requests
from typing import Optional, Dict, Any

class RetryConfig:
    MAX_RETRIES = 3
    BASE_DELAY = 2.0 # seconds
    BACKOFF_MULTIPLIER = 2.0

def call_with_retry(endpoint: str, method: str,
                    params: Dict[str, Any]) -> Dict[str, Any]:
    """Send MCP request with retry logic."""
    last_error = None

    for attempt in range(RetryConfig.MAX_RETRIES):
        try:
            response = requests.post(
                endpoint,
                json={
                    "jsonrpc": "2.0",
                    "method": method,
                    "params": params,
                    "id": 1
                },
                timeout=30
            )
            return response.json()

        except (requests.Timeout, requests.ConnectionError) as e:
            last_error = e
            if attempt < RetryConfig.MAX_RETRIES - 1:
                delay = RetryConfig.BASE_DELAY * \
                    (RetryConfig.BACKOFF_MULTIPLIER ** attempt)
                time.sleep(delay)

    return {
        "error": {
            "error_code": "E005",
            "error_description": f"Max retries exceeded: {last_error}"
        }
    }
```

כאשר שרת נכשל מספר פעמים, נמנע מניסיונות נוספים לתקופה מסוימת:

Circuit Breaker פשוט

```
from datetime import datetime, timedelta

class CircuitBreaker:
    def __init__(self, failure_threshold=5, reset_timeout=60):
        self.failures = 0
        self.threshold = failure_threshold
        self.reset_timeout = reset_timeout
        self.last_failure = None
        self.state = "CLOSED" # CLOSED, OPEN, HALF_OPEN

    def can_execute(self) -> bool:
        if self.state == "CLOSED":
            return True
        if self.state == "OPEN":
            if datetime.now() - self.last_failure > \
                timedelta(seconds=self.reset_timeout):
                self.state = "HALF_OPEN"
                return True
            return False
        return True # HALF_OPEN allows one try

    def record_success(self):
        self.failures = 0
        self.state = "CLOSED"

    def record_failure(self):
        self.failures += 1
        self.last_failure = datetime.now()
        if self.failures >= self.threshold:
            self.state = "OPEN"
```

5.10 תיעוד מובנה (Structured Logging)

הפרוטוקול מחייב תיעוד בפורמט JSON לצורך ניתוח וניפוי שגיאות. כל הודעת לוג חייבת לכלול את השדות הבאים:

טבלה 12: שדות חובה בהודעת לוג

Field	חובה	סוג	תיאור
timestamp	כן	ISO-8601	זמן האירוע
level	כן	string	DEBUG/INFO/WARN/ERROR
agent_id	כן	string	מזהה הסוכן
message_type	אל	string	סוג ההודעה
conversation_id	אל	string	מזהה השיחה
message	כן	string	תיאור האירוע
data	אל	object	נתונים נוספים

5.10.1 מימוש Logger

Logger מובנה

```
import json
import sys
from datetime import datetime
from typing import Optional, Dict, Any

class StructuredLogger:
    LEVELS = ["DEBUG", "INFO", "WARN", "ERROR"]

    def __init__(self, agent_id: str, min_level: str = "INFO"):
        self.agent_id = agent_id
        self.min_level = self.LEVELS.index(min_level)

    def log(self, level: str, message: str,
            message_type: Optional[str] = None,
            conversation_id: Optional[str] = None,
            data: Optional[Dict[str, Any]] = None):

        if self.LEVELS.index(level) < self.min_level:
            return

        log_entry = {
            "timestamp": datetime.now().isoformat(),
            "level": level,
            "agent_id": self.agent_id,
            "message": message
        }
```

63

if message_type:

log_entry["message_type"] = message_type

שימוש ב-Logger

```

logger = StructuredLogger("player:P01")

# Log received message
logger.info(
    "Received_game_invitation",
    message_type="GAME_INVITATION",
    conversation_id="conv-12345",
    data={"match_id": "R1M1", "opponent": "P02"}
)

# Log error
logger.error(
    "Failed_to_connect_to_referee",
    data={"endpoint": "http://localhost:8001", "error": "timeout"}
)

```

פלט הלוג:

פלט לוג

```

{"timestamp": "2025-01-15T10:30:00.123Z",
 "level": "INFO",
 "agent_id": "player:P01",
 "message": "Received_game_invitation",
 "message_type": "GAME_INVITATION",
 "conversation_id": "conv-12345",
 "data": {"match_id": "R1M1", "opponent": "P02"}}

```

5.11 אימות וטוקנים (Authentication)

החל מגרסה 2.1.0 של הפרוטוקול, כל הודעה חייבת לכלול `auth_token` לאימות. הטוקן מתקבל בעת הרישום ומשמש לזיהוי הסוכן בכל בקשה.

רישום וקבלת טוקן

```

import requests
from dataclasses import dataclass
from typing import Optional

@dataclass
class AgentCredentials:
    agent_id: str
    auth_token: str
    league_id: str

def register_player(league_endpoint: str,
                    player_info: dict) -> Optional[AgentCredentials]:
    """Register player and store auth token."""
    payload = {
        "jsonrpc": "2.0",
        "method": "register_player",
        "params": {
            "protocol": "league.v2",
            "message_type": "LEAGUE_REGISTER_REQUEST",
            "sender": f"player:{player_info['name']}",
            "player_meta": player_info
        },
        "id": 1
    }

    response = requests.post(league_endpoint, json=payload)
    result = response.json().get("result", {})

    if result.get("status") == "ACCEPTED":
        return AgentCredentials(
            agent_id=result["player_id"],
            auth_token=result["auth_token"],
            league_id=result["league_id"]
        )
    return None

```

בקשה עם אימות

```

class AuthenticatedClient:
    def __init__(self, credentials: AgentCredentials):
        self.creds = credentials

    def send_message(self, endpoint: str, message_type: str,
                    params: dict) -> dict:
        """Send authenticated message."""
        payload = {
            "jsonrpc": "2.0",
            "method": "mcp_message",
            "params": {
                "protocol": "league.v2",
                "message_type": message_type,
                "sender": f"player:{self.creds.agent_id}",
                "auth_token": self.creds.auth_token,
                "league_id": self.creds.league_id,
                **params
            },
            "id": 1
        }

        response = requests.post(endpoint, json=payload)
        return response.json()

```

טיפול בשגיאות אימות

```
def handle_auth_error(response: dict) -> bool:
    """Check for authentication errors."""
    error = response.get("error", {})
    error_code = error.get("error_code", "")

    if error_code == "E011": # AUTH_TOKEN_MISSING
        print("Error: auth_token is required")
        return False
    elif error_code == "E012": # AUTH_TOKEN_INVALID
        print("Error: auth_token is invalid or expired")
        # May need to re-register
        return False
    elif error_code == "E013": # REFEREE_NOT_REGISTERED
        print("Error: Referee must register first")
        return False

    return True # No auth error
```

5.12 בדיקות מקומיות

5.12.1 הרצה מקומית

הריצו כל סוכן בטרמינל נפרד:

הרצת הסוכנים

```
# Terminal 1: League Manager (start first)
python league_manager.py # Port 8000

# Terminal 2: Referee
python referee.py # Port 8001

# Terminal 3-6: Players
python player.py --port 8101
python player.py --port 8102
python player.py --port 8103
python player.py --port 8104
```

סדר הרצה חשוב:

1. קודם כל מנהל הליגה חייב לרוץ.

2. השופט נרשם למנהל הליגה בעת ההפעלה.

3. השחקנים נרשמים למנהל הליגה.

4. רק אז אפשר להתחיל את הליגה.

5.12.2 בדיקת חיבור

בדיקת שרת

```
import requests

def test_server(port):
    try:
        r = requests.post(
            f"http://localhost:{port}/mcp",
            json={"jsonrpc": "2.0", "method": "ping", "id": 1}
        )
        print(f"Port_{port}: OK")
    except:
        print(f"Port_{port}: FAILED")

# Test all servers
for port in [8000, 8001, 8101, 8102, 8103, 8104]:
    test_server(port)
```

5.13 טיפים למימוש

1. התחילו פשוט – ממשו קודם אסטרטגיה אקראית.
2. בדקו מקומית – הריצו ליגה עם עצמכם.
3. שמרו לוגים – תעדו כל הודעה.
4. טפלו בשגיאות – השתמשו ב-try/except.
5. עקבו אחר הפרוטוקול – השתמשו במבני JSON בדיוק.

6 דרישות תרגיל הבית

6.1 מטרת התרגיל

בתרגיל זה תממשו סוכן שחקן לליגת זוגי/אי-זוגי. בשלב זה, הסוכן שלכם ירוץ בסביבה שלכם בלבד. מומלץ לתאם עם סטודנטים אחרים כדי לוודא תאימות פרוטוקול. **חשוב מאוד:** השתמשו בפרוטוקול המוגדר במסמך זה בדיוק. אחרת הסוכן שלכם לא יוכל לתקשר עם אחרים.

חובה לבנות ולתכנן את הפרויקט בכפוף להנחיות של פרק 9 (פרוטוקול נתוני הליגה), פרק 10 (ערכת כלים בפיתוח), ופרק 11 (מבנה הפרויקט). כמו כן יש לוודא כי הפרויקט רץ ותפקד כמוגדר בפרק 8 (הרצת מערכת הליגה). תרגיל זה מבוסס על הספר:

סוכני בינה מלאכותית עם פרוטוקול הקשר המודל

AI Agents with Model Context Protocol

מאת ד"ר יורם סגל

December 9, 2025

לכן מאוד מומלץ לקרוא וללמוד את הנושא לעומק.

6.2 משימות חובה

6.2.1 משימה 1: מימוש סוכן שחקן

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

1. `handle_game_invitation` – קבלת הזמנה למשחק והחזרת `GAME_JOIN_ACK`.
2. `choose_parity` – בחירת "זוגי" או "אי-זוגי" והחזרת `CHOOSE_PARITY_RESPONSE`.
3. `notify_match_result` – קבלת תוצאת משחק ועדכון מצב פנימי.

6.2.2 משימה 2: רישום לליגה

הסוכן חייב לשלוח בקשת רישום למנהל הליגה. הבקשה תכלול:

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

6.2.3 משימה 3: בדיקה עצמית

לפני ההגשה, בדקו את הסוכן שלכם:

1. הריצו ליגה מקומית עם 4 שחקנים.

2. ודאו שהסוכן מגיב לכל סוג הודעה.
3. ודאו שתבניות ה-JSON תואמות לפרוטוקול.

6.3 דרישות טכניות

6.3.1 שפת תכנות

אתם יכולים לבחור כל שפה שתמצאו. העיקר שהסוכן:

- מממש שרת HTTP.
- מגיב לבקשות 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 קריסות.
- לטפל בשגיאות קלט.
- לא להפסיק לפעול באמצע ליגה.

6.4 תהליך העבודה

6.4.1 שלב 1: פיתוח מקומי

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

6.4.2 שלב 2: ליגה פרטית

1. הריצו ליגה מקומית עם 4 עותקים של הסוכן.
2. בדקו שכל התקשורת עובדת.
3. שפרו את האסטרטגיה (אופציונלי).

6.4.3 שלב 3: בדיקת תאימות עם סטודנטים אחרים

1. תאמו עם סטודנט אחר להחלפת סוכנים.
2. בדקו שהסוכנים מתקשרים ביניהם כראוי.
3. ודאו שתבניות ה-JSON תואמות לפרוטוקול.

6.4.4 מבט לעתיד: ליגת כיתה

הערה חשובה

בעתיד, ייתכן שתדרשו:

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

נושא זה טרם נסגר וייתכנו שינויים. עליכם 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 הכולל:
- תיאור מלא של הארכיטקטורה והמימוש.
 - תיאור האסטרטגיה שנבחרה והסיבות לבחירה.
 - קשיים שנתקלתם בהם והפתרונות שמצאתם.
 - תיעוד של תהליך הפיתוח והבדיקות.
 - מסקנות מהתרגיל והמלצות לשיפור.

6.5.2 פורמט הגשה

יש להגיש קישור לריפוזיטורי כשהוא ציבורי. ויש להגיש בנוהל ההגשה הרגיל כפי שהוגשו התרגילים הקודמים.

6.6 דגשים כלליים לבדיקת העבודה

מעבר לדרישות הרגילות, הקריטריונים הבאים ייבדקו:

טבלה 13: קריטריונים לבדיקה

קריטריון	תיאור
תפקוד בסיסי	הסוכן עובד, עונה להודעות, משחק במשחקים
תאימות פרוטוקול	תבניות JSON תואמות בדיוק לפרוטוקול
יציבות	הסוכן יציב, לא קורס, מטפל בשגיאות
איכות קוד	קוד נקי, מתועד, מאורגן
תיעוד	הוראות הרצה ברורות, תיאור מפורט
אסטרטגיה	מימוש אסטרטגיה מעניינת (לא רק אקראי)

6.7 שאלות נפוצות

6.7.1 האם אפשר להשתמש בספריות חיצוניות?

כן. אתם יכולים להשתמש בכל ספרייה שתמצאו. ודאו שאתם מספקים הוראות התקנה.

6.7.2 האם חייבים להשתמש ב-Python?

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

6.7.3 מה קורה אם הסוכן שלי קורס?

הסוכן יספוג 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 שאתם מגישים.

6.7.5 איך אדע מה הדירוג שלי?

טבלת הדירוג תפורסם לאחר כל מחזור. תוכלו לראות את המיקום של הסוכן שלכם.

6.8 סיכום

1. ממשו סוכן שחקן שעומד בפרוטוקול.
2. בדקו מקומית לפני הגשה.
3. הגישו את הקוד והדוח.
4. הסוכן שלכם ישחק בליגת הכיתה.

בהצלחה!

מידע נוסף:

- לשאלות ובירורים פנו לד"ר יורם סגל.
- מומלץ לקרוא את הספר "סוכני AI עם MCP" [1].
- 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 מהווה 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) בליגה הוא באמת סוכן AI? התשובה היא חד-משמעית: כן.

סוכן AI מוגדר כישות המקיימת 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, ומחליטה בעצמה מה לבצע על בסיס המצב הנוכחי.

7.1.2 ארבעת המאפיינים של סוכן AI

נבחן את סוכן השחקן בליגה 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 הזמנה למשחק (GAME_INVITATION), בקשות לבחירת זוגיות (CHOOSE_PARITY_CALL), ותוצאות משחקים (GAME_OVER) מהשופט ומנהל הליגה.
 3. **פעולה** – הסוכן משפיע על הסביבה. השחקן מבצע פעולות על ידי שליחת בחירות (CHOOSE_PARITY_RESPONSE) ואישורי הגעה (GAME_JOIN_ACK) למשחקים.
 4. **תכליתיות** – יש לו מטרה מוגדרת. מטרתו היא לשחק, לנצח במשחקים ולעדכן את מצבו הפנימי, כגון היסטוריית ניצחונות והפסדים.
- סוכן השחקן יכול אף להשתמש במודל שפה גדול (LLM) כדי לבחור את האסטרטגיה הטובה ביותר. בכך הוא מדגים "חשיבה" או "הסקת מסקנות" לפני ביצוע הפעולה.

7.2 השחקן בארכיטקטורת MCP

7.2.1 שרת או לקוח?

בארכיטקטורת ליגת זוגי/אי-זוגי, השחקן הוא בעיקרו **שרת MCP**. שרת MCP הוא הרכיב שחושף יכולות ושירותים, המכונים "כלים" (Tools), "משאבים" (Resources) או "הנחיות" (Prompts). השרת מוגדר כתהליך נפרד הפועל על פורט מוגדר ומספק "שער" לעולם החיצון [2].

סוכן השחקן נדרש לממש שרת HTTP שמקבל בקשות POST בנתיב /mcp. הכלים שהוא חושף נקראים באמצעות פרוטוקול JSON-RPC 2.0. הכלים שהשחקן מחויב לממש 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 כלקוחות או אורקסטרטורים (Orchestrators).

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

לסיכום: אף על פי שסוכן השחקן הוא סוכן AI אוטונומי, מבחינת מימוש פרוטוקול MCP, הוא ממלא את תפקיד השרת המציע יכולות לאורקסטרטורים המרכזיים.

7.3 השופט ומנהל הליגה כסוכני AI

7.3.1 סוכנים בדרגה 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. זוהי מהות האוטונומיות והתכליתיות של סוכן AI.

7.3.2 שרתי MCP שפועלים גם כלקוחות

שני הסוכנים הללו מוגדרים כשרתי MCP:

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

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

טבלה 14: מאפייני סוכן IA עבור השופט ומנהל הליגה

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

- השופט חייב לפעול כלקוח כדי להירשם למנהל הליגה (REFEREE_REGISTER_RE-QUEST).

- השופט חייב לפעול כלקוח כדי לקרוא לכלי choose_parity של סוכן השחקן.
 - מנהל הליגה חייב לפעול כלקוח כדי לשלוח את הכרזות המחזור לסוכני השחקנים.
 במערכת זו, השרתים המרכזיים הם למעשה לקוחות-אורקסטרטורים כאשר הם צריכים להניע פעולה אצל שרתי השחקנים.

7.4 היפוך התפקידים: תובנה מרכזית

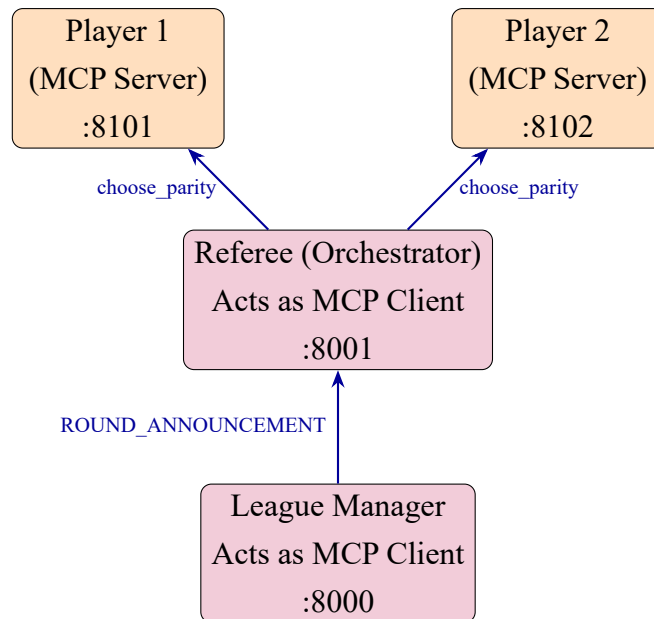
7.4.1 הפרדיגמה המסורתית

בארכיטקטורת שרת-לקוח הטיפוסית, הלקוח הוא הרכיב האקטיבי ששולח בקשות, והשרת הוא הרכיב הפסיבי שמחכה לבקשות. בליגת ה-AI, מתקיים היפוך תפקידים יצירתי.

7.4.2 היפוך התפקידים בליגה

השחקן (הסוכן האוטונומי) הוא השרת: למרות שהשחקן הוא הישות האוטונומית שצריכה לבצע פעולה, הוא נדרש לחשוף את יכולותיו כשרת MCP.

השופט ומנהל הליגה (האורקסטרטורים) הם הלקוחות: השופט הוא האורקסטרטור שפועל כלקוח 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** (מנוהלת על ידי מנהל הליגה) – גיוס שחקנים, לוח משחקים (Round-Robin), וטבלת דירוג.
2. **שכבת השיפוט** (מנוהלת על ידי השופט) – ניהול משחק בודד ואימות מהלכים.
3. **שכבת חוקי המשחק** (מנוהלת על ידי מודול נפרד) – הלוגיקה הספציפית למשחק זוגי/אי-זוגי.

7.5.2 היתרון של ההפרדה

השחקן, בכך שהוא חושף ממשק MCP סטנדרטי (JSON-RPC 2.0 על גבי HTTP), מאפשר לליגה להישאר אגנוסטית לשפת הפיתוח או לאסטרטגיה הפנימית שלו. זהו פתרון לבעיית הפרגמנטציה שבה לכל סוכן ולכל מודל 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 עם 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 השרת

7.6.1 הדילמה

עולה שאלה מעניינת: מצד אחד, השחקן מוגדר כשרת MCP שחושף יכולות. מצד שני, הוא מתואר כסוכן AI אוטונומי שיכול להשתמש ב-LLM כ"מוח" לבחירת אסטרטגיה. בהגדרות מסורתיות, שרת אינו מפעיל "מוח" אלא ממלא בקשה.

7.6.2 הפתרון: הפרדת תפקידים

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

הסוכן הוא גם שרת וגם לקוח (בפועל): כל אחד מהסוכנים הוא בפועל גם שרת וגם לקוח. תפקיד השרת נדרש לכל סוכן המארח את עצמו כדי לאפשר לסוכנים אחרים לקרוא לכליו. תפקיד הלקוח נדרש לכל סוכן שצריך ליזום אינטראקציה.

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

כאשר השרת מקבל בקשת `choose_parity`:

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

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

7.6.3 אנלוגיה: תחנת שירות לקוחות

ניתן לדמיין את הארכיטקטורה כתחנת שירות לקוחות:

- MCP (פרוטוקול) – הוא הטלפון והשפה שבה מדברים (JSON-RPC על HTTP).
 - השחקן (שרת) – הוא משרד השירות עם קו טלפון משלו.
 - האסטרטגיה/LLM (המוח) – הוא היועץ החכם היושב בתוך המשרד, שמקבל את השיחה, מחשב את המענה, ומכתיב לשכבת ה-MCP איזו תשובה לשלוח בחזרה.
- הכלים הפנימיים (ה-LLM והלוגיקה) אינם חשופים ישירות לפרוטוקול 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 אסטרטגית, המנהל את שכבת הליגה. הוא אינו מעורב בחוקי המשחק עצמם, אלא בניהול הכללי: לוח משחקים וטבלת דירוג.

יתרון ההפרדה: אם הליגה תרצה להחליף את המשחק מזוגי/אי-זוגי לאיקס-עיגול (Tic-Tac-Toe), מנהל הליגה כמעט ולא ישתנה. זוהי הדגמה מושלמת של עקרון הפרדת התפקידים שמקדם ה-MCP.

7.7.2 השופט – המיישם הדינמי

השופט מגלם את שכבת השיפוט. הוא אינו יודע את כללי המשחק (שמטופלים על ידי מודול נפרד), אלא הוא אחראי על ניהול השיחה (Conversation Lifecycle) בין השחקנים. השופט מוודא שהשחקנים עומדים במועדי התגובה (Deadlines). הוא זה שמפעיל את לולאת הסוכן החיצונית עבור השחקנים – הוא קורא לכלי `choose_parity` שלהם ובכך מניע את הפעולה האוטונומית של השחקן.

MCP מאפשר את חלוקת התפקידים הברורה: השופט ומנהל הליגה אחראים על ה"איך" (הפרוטוקול והתקשורת), בעוד השחקנים אחראים על ה"מה" (האסטרטגיה 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 הודעות מהמערכת.
- פעולה – השחקן שולח תגובות.
- תכליתיות – השחקן שואף לנצח.

7.8.2 עקרונות יסוד של MCP

התרגיל מלמד את עקרונות הליבה של פרוטוקול MCP:

1. **ממשק סטנדרטי** – כל סוכן חושף כלים דרך JSON-RPC 2.0.
2. **הפרדת תפקידים** – שכבת הליגה, שכבת השיפוט, ושכבת חוקי המשחק.
3. **אגנוסטיות לשפה** – ניתן לממש 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 חוויית הלמידה

בסיום התרגיל, הסטודנט יבין:

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

7.9 סיכום

תרגיל ליגת זוגי/אי-זוגי מהווה מודל פדגוגי מושלם להבנת פרוטוקול MCP וסוכני AI. המשחק הפשוט מאפשר להתמקד בעקרונות הארכיטקטוניים מבלי להסתבך בלוגיקה מורכבת.

הסטודנט לומד שסוכן AI יכול להיות גם שרת MCP – היפוך תפקידים יצירתי המאפשר לאורקסטרטור לקרוא לסוכנים ולהניע את פעולתם. ההפרדה לשכבות מבטיחה שניתן להחליף את משחק הליגה בעתיד מבלי 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 ליגה עם מנהל ליגה אחד, שני שופטים וארבעה שחקנים. הדוגמאות מבוססות על פרוטוקול league.v2 המתואר בפרקים הקודמים.

8.1 תצורת המערכת

8.1.1 פורטים וטרמינלים

כל סוכן במערכת פועל כשרת HTTP נפרד על פורט ייעודי ב-localhost. בדוגמה זו נריץ 7 טרמינלים:

טבלה 15: הקצאת 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 תפקידי האורקסטרטורים

במערכת שני סוגי Orchestrator:

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

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

8.2 סדר הפעלה

8.2.1 עקרון סדר ההפעלה

סדר ההפעלה קריטי לתפקוד תקין של המערכת:

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

8.2.2 טרמינל 1 – מנהל הליגה

הפעלת מנהל הליגה

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

מנהל הליגה מאזין לבקשות POST בכתובת `http://localhost:8000/mcp`.

8.2.3 טרמינלים 2-3 – שופטים

הפעלת 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-
EREE_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 הלקוח למנהל הליגה.

8.3.1 בקשת רישום שופט**REFEREE_REGISTER_REQUEST – בקשת 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_αAlpha",
      "version": "1.0.0",
      "game_types": ["even_odd"],
      "contact_endpoint": "http://localhost:8001/mcp",
      "max_concurrent_matches": 2
    }
  },
  "id": 1
}
```

REFEREE_REGISTER_RESPONSE – תגובת 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".referee_id:

8.4 שלב 2: רישום שחקנים

לאחר שהשופטים נרשמו, כל שחקן שולח בקשת רישום.

8.4.1 בקשת רישום שחקן

LEAGUE_REGISTER_REQUEST – בקשת 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": "AgentֶAlpha",
      "version": "1.0.0",
      "game_types": ["even_odd"],
      "contact_endpoint": "http://localhost:8101/mcp"
    }
  },
  "id": 1
}
```

LEAGUE_REGISTER_RESPONSE – תגובת 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
}
```

באופן דומה:

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

מנהל הליגה שומר מפה `player_id` → `contact_endpoint` ומפה `referee_id` → `contact_endpoint`.

8.5 שלב 3: יצירת לוח משחקים

לאחר שכל השחקנים והשופטים נרשמו, מנהל הליגה מפעיל לוגיקת `create_schedule` 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 הודעת 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 התחיל לוגית. המשחקים עצמם מתחילים רק כאשר השופט מזמין את המשתתפים.

8.7 שלב 5: ניהול משחק בודד

נתאר 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:

הזמנה למשחק – GAME_INVITATION ל-P01

```
{
  "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-rlm1-001",
    "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:

P02-5 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-rlm1-001",
    "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 5.2: אישורי הגעה

כל שחקן מחזיר GAME_JOIN_ACK תוך 5 שניות.

אישור מ-P01:

אישור הגעה - GAME_JOIN_ACK מ-P01

```
{
  "jsonrpc": "2.0",
  "result": {
    "protocol": "league.v2",
    "message_type": "GAME_JOIN_ACK",
    "sender": "player:P01",
    "timestamp": "2025-01-15T10:15:01Z",
    "conversation_id": "conv-rlm1-001",
    "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:

אישור הגעה - GAME_JOIN_ACK מ-P02

```
{
  "jsonrpc": "2.0",
  "result": {
    "protocol": "league.v2",
    "message_type": "GAME_JOIN_ACK",
    "sender": "player:P02",
    "timestamp": "2025-01-15T10:15:02Z",
    "conversation_id": "conv-rlm1-001",
    "auth_token": "tok-p02-def456",
    "match_id": "R1M1",
    "player_id": "P02",
    "arrival_timestamp": "2025-01-15T10:15:02Z",
    "accept": true
  },
  "id": 1002
}
```

כאשר השופט קיבל שני ACK חיוביים בזמן המותר, הוא מעביר את מצב המשחק ל-COLLECTING_CHOICES.

השופט שולח CHOOSE_PARITY_CALL לכל שחקן.

בקשת בחירה ל-P01:

בקשת בחירה - CHOOSE_PARITY_CALL ל-P01

```
{
  "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
}
```

תגובת P01 (בחר "even"):

תגובת בחירה - P01-מ 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-rlm1-001",
    "auth_token": "tok-p01-xyz789",
    "match_id": "R1M1",
    "player_id": "P01",
    "parity_choice": "even"
  },
  "id": 1101
}
```

תגובת P02 (בחר "odd"):

תגובת בחירה - P02-מ 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-rlm1-001",
    "auth_token": "tok-p02-def456",
    "match_id": "R1M1",
    "player_id": "P02",
    "parity_choice": "odd"
  },
  "id": 1102
}
```

כאשר שתי הבחירות התקבלו תקין ובזמן, השופט מעביר את מצב המשחק ל-DRAWING_NUMBER.

8.7.4 שלב 5.4: הגרלת מספר וקביעת מנצח

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

$\text{drawn_number} = 8$ -

number_parity = "even" -
בחירת P01 = "even" → צדק -
בחירת P02 = "odd" → טעה -
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 (סטטיסטיקות, היסטוריה) ומחזיר תשובה כללית.

השופט מדווח MATCH_RESULT_REPORT לליגה:

MATCH_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 מעדכן טבלת נקודות בהתאם לטבלת הניקוד (ניצחון = 3 נקודות).

8.8 שלב 6: סיום מחזור ועדכון דירוג

מחזור מספר 1 מסתיים כאשר לכל משחקי המחזור התקבל MATCH_RESULT_REPORT.
מנהל הליגה:

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

LEAGUE_STANDINGS_UPDATE - עדכון דירוג

```
{
  "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_Alpha",
        "played": 1,
        "wins": 1,
        "draws": 0,
        "losses": 0,
        "points": 3
      },
      {
        "rank": 2,
        "player_id": "P03",
        "display_name": "Agent_Gamma",
        "played": 1,
        "wins": 0,
        "draws": 1,
        "losses": 0,
        "points": 1
      },
      {
        "rank": 3,
        "player_id": "P04",
        "display_name": "Agent_Delta",
        "played": 1,
        "wins": 0,
        "draws": 1,
        "losses": 0,
        "points": 1
      },
      {
        "rank": 4,
        "player_id": "P02",
        "display_name": "Agent_Beta",
        "played": 1,
        "wins": 0,
        "draws": 0,
        "losses": 1,

```

לאחר שליחת עדכון הדירוג, מנהל הליגה שולח הודעת ROUND_COMPLETED לסמן את סיום המחזור:

סיום מחזור - 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 המחזורים, מנהל הליגה שולח הודעת LEAGUE_COMPLETED:

LEAGUE_COMPLETED – סיום ליגה

```
{
  "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": "Agent Alpha",
      "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 הודעת שגיאה מתאימה.

LEAGUE_ERROR – שגיאת 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
}
```

GAME_ERROR – שגיאת משחק

```

{
  "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 כלי שאילתות זמינים

המסמך מגדיר כלי MCP גנריים שכל סוכן יכול לחשוף לצרכי דיבוג ובירור.

8.11.1 שאילתת דירוג ממנהל הליגה

שחקן שרוצה לוודא את דירוגו קורא למנהל הליגה:

LEAGUE_QUERY – שאילתת דירוג

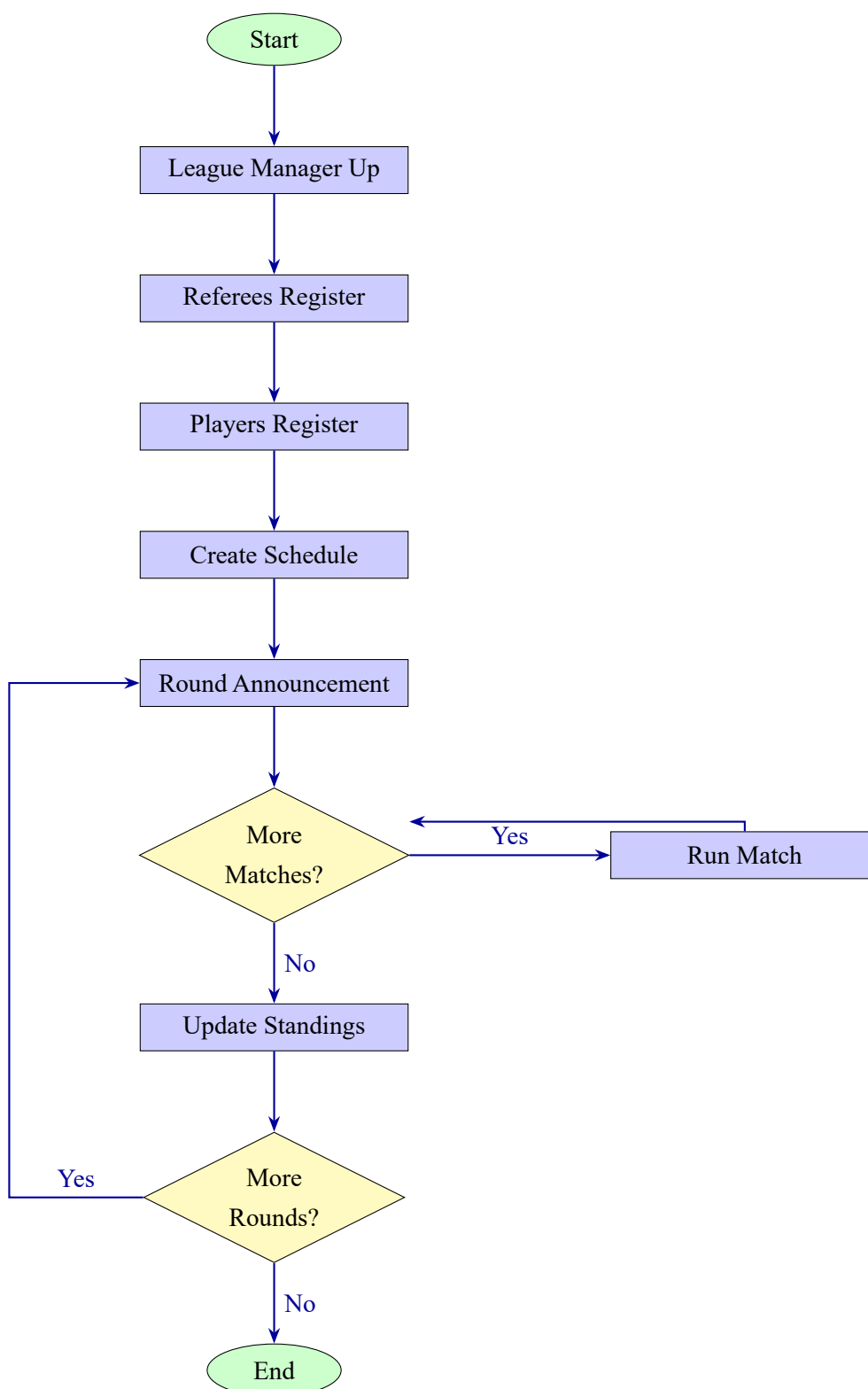
```
{
  "jsonrpc": "2.0",
  "method": "league_query",
  "params": {
    "protocol": "league.v2",
    "message_type": "LEAGUE_QUERY",
    "sender": "player:P01",
    "timestamp": "2025-01-15T10:25:00Z",
    "conversation_id": "conv-query-standings-001",
    "auth_token": "tok-p01-xyz789",
    "league_id":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

מנהל הליגה מחזיר `result` שכולל אובייקט `standings` בפורמט זה ל-`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 דיבוג).
- **כלי בשחקן:** `get_player_state` – נותן את היסטוריית המשחקים.

8.12 דיאגרמת זרימה מלאה



8.13 טבלת תפקידי הסוכנים

טבלה 17: תפקידי הסוכנים במערכת

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

8.14 סיכום

נספח זה הציג את:

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

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

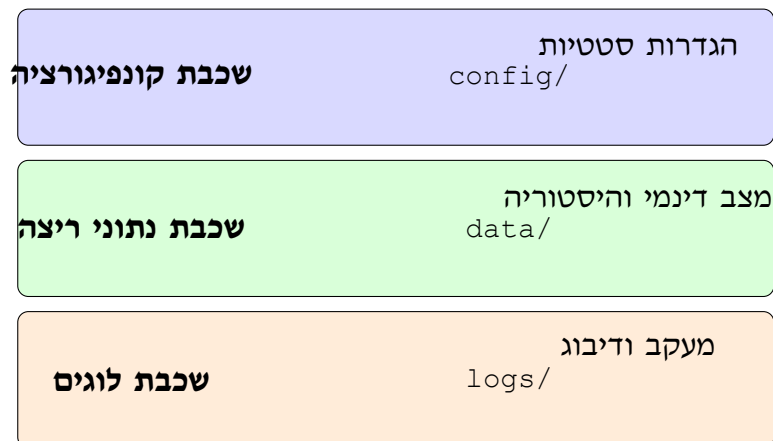
9 פרוטוקול נתוני הליגה

9.1 מבוא: הקוד הגנטי של חברת הסוכנים

כאשר אנו בונים חברה של סוכנים אוטונומיים – שחקנים, שופטים ומנהלי ליגה – אנו למעשה יוצרים תרבות דיגיטלית חדשה. כמו בכל חברה אנושית, גם כאן נדרשים שלושה יסודות קריטיים:

1. **חוקים משותפים** – הפרוטוקול שהגדרנו בפרקים הקודמים.
 2. **זיכרון קולקטיבי** – היכולת לשמור ולשחזר מידע היסטורי.
 3. **קוד גנטי** – הקונפיגורציה שמגדירה 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 מנחים

כל קובץ במערכת עומד בעקרונות הבאים:

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

9.3 שכבת הקונפיגורציה – config/

שכבה זו מכילה את ה"קוד הגנטי" של המערכת – הגדרות 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 ברירת המחדל עבור:
- הגדרות רשת (network) – פורטים וכתובות.
 - הגדרות אבטחה (security) – טוקנים ו-TTL.
 - זמני המתנה (timeouts) – תואמים להגדרות הפרוטוקול בפרק 2.
 - מדיניות ניסיון חוזר (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 אלפי סוכנים.
 - **משתמשים:** מנהל הליגה, כלי Deployment.
 - **מיקום:** SHARED/config/agents/agents_config.json
- קובץ זה מכיל את "ספר האזרחים" של חברת הסוכנים:
- league_manager – פרטי מנהל הליגה.
 - referees[] – רשימת כל השופטים הרשומים.
 - players[] – רשימת כל השחקנים הרשומים.

9.3.3 קונפיגורציית ליגה – config/leagues/<league_id>.json

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

דוגמה: קונפיגורציית 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 (לטעינת מודול חוקים), מנהל ליגה.
 - **מיקום:** SHARED/config/games/games_registry.json
- המערכת תומכת בסוגי משחקים מרובים. כל משחק מגדיר:
- game_type – מזהה ייחודי.
 - rules_module – מודול החוקים לטעינה.
 - max_round_time_sec – זמן מקסימלי לסיבוב.

9.3.5 ברירות מחדל לסוכנים – config/defaults/

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

9.4 שכבת נתוני הריצה – data/

אם שכבת הקונפיגורציה היא ה"קוד הגנטי", שכבת נתוני הריצה היא ה"זיכרון ההיסטורי" של החברה. כאן נשמרים כל האירועים שקורים 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 היסטוריית מחזורים – data/leagues/<league_id>/rounds.json

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

9.4.3 נתוני 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:

- lifecycle – מצב המשחק וזמנים.

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

9.4.4 היסטוריית שחקן – data/players/<player_id>/history.json

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

דוגמה: היסטוריית שחקן

```
{
  "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 מרכזי – logs/league/<league_id>/league.log.jsonl

- **פורמט:** JSON Lines (כל שורה אובייקט JSON נפרד).
- **משתמשים:** DevOps, תמיכה 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 בנספח זה זמינים בתיקיה המשותפת:

L07/SHARED/

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

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

טבלה 18: סיכום קובצי הקונפיגורציה והנתונים

שכבה	נתיב	מטרה	משתמש
קונפיג	config/system.json	פרמטרים גלובליים	כל הסוכנים
קונפיג	config/agents/	רישום סוכנים	מנהל ליגה
קונפיג	config/leagues/	הגדרות ליגה	מנהל ליגה
קונפיג	config/games/	רישום משחקים	שופטים
קונפיג	config/defaults/	ברירות מחדל	סוכנים
ריצה	data/.../stand-ings.json	טבלת דירוג	כולם
ריצה	data/.../rounds.json	היסטוריית מחזורים	מנהל ליגה
ריצה	data/matches/	פרטי משחק	אנליטיקה
ריצה	data/players/	היסטוריה אישית	שחקן
לוגים	logs/league/	לוג מרכזי	DevOps
לוגים	logs/agents/	לוג סוכן	מפתחים

9.8 סיכום

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

כמו בחברה אנושית, גם כאן:

- **הקונפיגורציה** היא ה"חוקה" – הכללים היסודיים שכולם מכירים.

- **נתוני הריצה** הם ה"ארכיון ההיסטורי" – הזיכרון הקולקטיבי.

- **הלוגים** הם ה"עיתונות" – תיעוד מה שקורה בזמן אמת.

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

10 ערכת כלים בפייתון

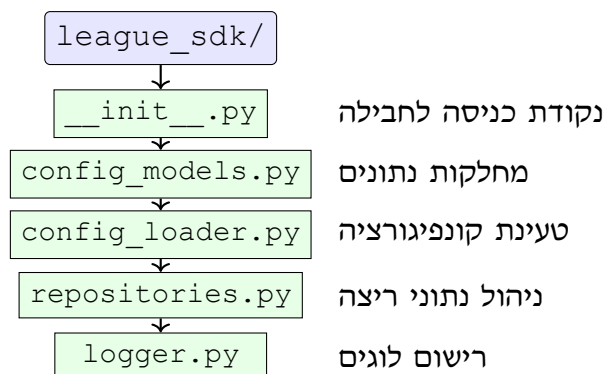
10.1 מבוא: מהקונפיגורציה לקוד

בנספח ב הצגנו את ארכיטקטורת הנתונים מבוססת קובצי JSON – שלוש שכבות של קונפיגורציה, נתוני ריצה, ולוגים. אך כיצד סוכן בודד ניגש לנתונים אלו בפועל? נספח זה מציג את league_sdk – ספריית Python שמגשרת בין קובצי ה-JSON לבין האובייקטים שבהם משתמשים הסוכנים. הספרייה מיישמת שני דפוסי עיצוב מרכזיים:

1. **Dataclasses** – מודלים טיפוסיים (typed models) שמשקפים את מבנה ה-JSON.

2. **Repository Pattern** – שכבת הפשטה לגישה לנתונים.

10.2 מבנה הספרייה



10.3 מודלים טיפוסיים – config_models.py

10.3.1 הגישה: Dataclasses

Python 3.7+ מספקת את הדקורטור @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 קונפיגורציית מערכת

הקובץ מגדיר את כל המודלים התואמים ל-`config/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
```

כל סוג סוכן מקבל מחלקת 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
```

הגדרות ספציפיות לליגה 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 טוען קונפיגורציה – ConfigLoader

10.4.1 העיקרון: טעינה עצלה עם מטמון

המחלקה ConfigLoader מיישמת את דפוס ה-Lazy Loading – קובצי הקונפיגורציה נטענים רק כשיש בהם צורך, ונשמרים במטמון לגישה חוזרת:

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

ConfigLoader מספקת ממשק אחיד לטעינת כל סוגי הקונפיגורציה:

טבלה 19: שיטות הטעינה של ConfigLoader

שיטה	מחזירה	תיאור
load_system()	SystemConfig	קונפיגורציה גלובלית
load_agents()	AgentsConfig	רשימת כל הסוכנים
load_league(id)	LeagueConfig	קונפיגורציית ליגה 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:
        """Load standings from JSON file."""
        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:
        """Save standings to JSON file."""
        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):
        """Update a player's standings after a match."""
        standings = self.load()
        # ... update logic
        self.save(standings)
```

10.5.3 מאגרים נוספים

הספרייה כוללת מאגרים נוספים לניהול נתוני ריצה:

טבלה 20: מאגרי נתונים זמינים

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

10.6 רישום לוגים – JsonLogger

10.6.1 פורמט JSON Lines

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

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

10.6.2 מחלקת הלוגר

מחלקת 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")
```


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)
```

שימוש ב-ConfigLoader במנהל ליגה

```
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

```

TIMEOUT שגיאת

```

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 שכבת הפשטה 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 נתוני ריצה בדפוס Repository.
 - logger.py – מאפשר רישום לוגים מובנה בפורמט JSONL.
- שימוש בספרייה זו מבטיח:

1. **עקביות** – כל הסוכנים משתמשים באותם מודלים ונתונים.
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:

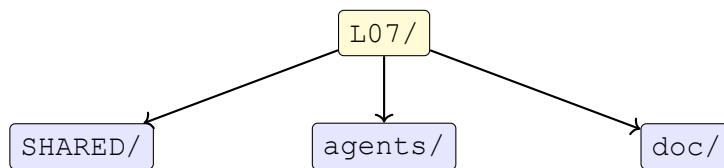
L07/SHARED/league_sdk/

11 מבנה הפרויקט

11.1 מבוא: מפת הדרכים

לאחר שהכרנו את הפרוטוקול, הודעות 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 ומוכן.

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 בתיקיה נפרדת
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 תיקיית 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 תיקיית התיעוד – doc/

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

טבלה 22: קבצי סוכן טיפוס

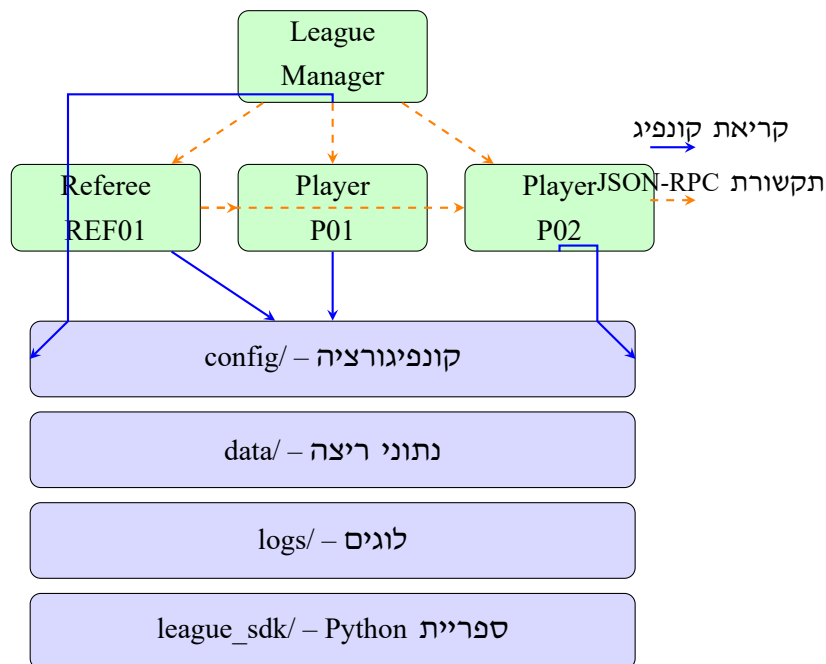
קובץ	תפקיד
main.py	נקודת כניסה - אתחול השרת וטעינת קונפיגורציה
handlers.py	טיפול בהודעות נכנסות לפי סוג
strategy.py	לוגיקת קבלת החלטות (לשחקנים)
game_logic.py	חוקי המשחק (לשופטים)
requirements.txt	תלויות Python

```

|   └─ 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 קריאה וכתיבה

טבלה 23: הרשאות גישה לקבצים

הערות	כותב	קורא	קובץ/תיקייה
קריאה בלבד לסוכנים	מנהל מערכת	כולם	config/*
עדכון משחק אחרי	מנהל ליגה	כולם	standings.json
היסטוריית מחזורים	מנהל ליגה	כולם	rounds.json
קובץ למשחק	שופט	כולם	matches/*.json
היסטוריה אישית	השחקן	השחקן	history.json
לוג משלו בלבד	כל סוכן	DevOps	logs/*

11.8 התקנה והפעלה

11.8.1 דרישות מקדימות

- Python 3.10+
- pip או poetry לניהול חבילות
- גישה לתיקיית 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. הפרדה ברורה – כל רכיב בתיקייה משלו.
2. משאבים משותפים – SHARED/ מרכז את כל הנתונים.
3. עצמאות סוכנים – כל סוכן יכול לפעול באופן עצמאי.
4. תיעוד מובנה – דוגמאות ומפרטים ב-doc/.
5. יכולת הרחבה – קל להוסיף סוכנים וליגות חדשות.

הקבצים המלאים זמינים בתיקייה:

L07/SHARED/

מומלץ לשכפל את המבנה ולהתחיל לבנות את הסוכנים שלכם!

- 1 Y. Segal, *AI Agents with MCP*. Dr. Yoram Segal, 2025, Hebrew edition.
- 2 Anthropic, *Model context protocol specification*, 2024. [Online]. Available: <https://modelcontextprotocol.io/>
- 3 JSON-RPC Working Group, *Json-rpc 2.0 specification*, 2010. [Online]. Available: <https://www.jsonrpc.org/specification>
- 4 K. Stratis, *AI Agents with MCP*. O'Reilly Media, 2025, Early Release.