

תרגיל בית: ליגת זוגי/אי-זוגי

Homework Exercise: Even/Odd League

ד"ר יורם סגל

כל הזכויות שמורות - © Dr. Segal Yoram

2025

גרסה 2.0

תוכן העניינים

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

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

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

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

51	מדריך מימוש	5
51	ארQUITECTURA ככלית	5.1
51	תרשים רכיבים	5.1.1
51	תפקיד ה-Orchestrator	5.1.2
52	מימוש שירות 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 לliga	5.4.2
55	לוגיקת קביעת מנצח	5.4.3
55	מימוש מנהל liga	5.5
55	כלים נדרשים	5.5.1

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

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

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

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

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

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

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

133

12 **מקורות**

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. החלטה 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 משתמש בעקרונות אלה בתרגום.

1.2 פרוטוקול MCP – Model Context Protocol

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

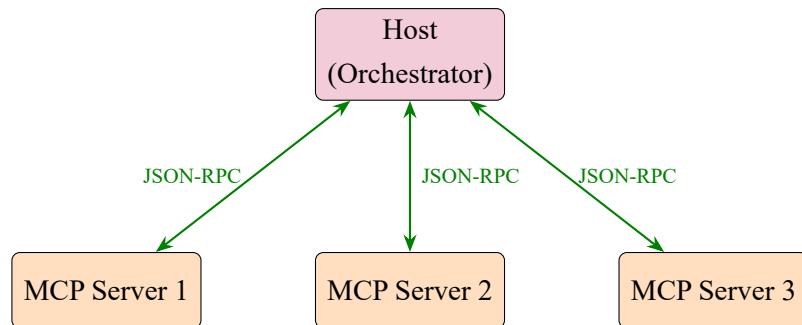
1.2.1 עקרונות הפרוטוקול

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 (Tools)** – 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. השירות חושף "כלים" שאפשר לקרוא להם. כל כלי הוא פונקציה עם פרמטרים מוגדרים.
מארח (Host) – רכיב שמתאים בין שירותים. המארח שולח בקשות לשירותים. הוא מקבל תשובות ו מעבד אותן.



1.3 תחבורה HTTP על localhost

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

1.3.1 הגדרת פורטים

נדיר פורטים קבועים לכל סוכן:

8000 – פורט League Manager -

8001 – פורט Referee -

8104 עד 8101 – פורטים שחכים -

כל סוכן ממש שרת HTTP פשוט. השירות מקבל בקשות POST בנתיב `mcp/.
הבקשה היא MCP 2.0 JSON-RPC`.

1.3.2 דוגמה לכתובת סוכן

כתובת שרת League Manager:

`http://localhost:8000/mcp`

כתובת שרת שחון ראשון:

`http://localhost:8101/mcp`

1.4 מבנה הודעת JSON-RPC

כל הודעה בפרוטוקול היא אובייקט JSON. הודעה 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 "0.2".
- method – שם הכלים שורוצים להפעיל.
- params – פרמטרים לכלוי.
- id – מזהה ייחודי לבקשתו.

1.5 מטרת התרגיל

בתרגיל זה נבנה מערכת לигה לsoccer AI. המערכת תכלול שלושה סוגי סוכנים:

1. **מנהל ליגה** (League Manager) – מנהל את הליגה, כולל רישוםשחקנים ושופטים.
2. **שופט** (Referee) – נרשם למנהל הליגה ומנהל משחקים בודדים.
3. **סוכני שחקן** (Player Agents) – משתמשים במשחקים.

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

1.5.1 יעדי הלמידה

בסיום התרגיל תוכלו:

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

- לוודא תאימות פרוטוקול עם סטודנטים אחרים.

חשיבות: כל הסטודנטים ישתמשו באותו פרוטוקול. זה אפשר לסוכנים שלכם לשחק זה נגד זה בעתיד.

2 פרוטוקול הליגה הכללי

2.1 עקרונות הפרוטוקול

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

2.1.1 הפרדה לשלווש שכבות

המערכת מורכבת משלווש שכבות:

1. **שכבת הליגה** – ניהול טורניר, רישום שחknim, טבלת דירוג.
2. **שכבת השיפוט** – ניהול משחק בודד, אימותות מהלכים, הכרזת מנצח.
3. **שכבת חוקי המשחק** – לוגיקת משחק ספציפית (זוגי/אי-זוגי, איקס-עיגול, ועוד').

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 שחknim 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 לשפטו משחקים, הוא חייב להירשם אצל מנהל הליגה.

השופט אחראי על:

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

השופט פועל כשרת MCP על פורט 8001. יכולם 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 (8104-8101).

2.3 מזהים בפרוטוקול

כל רכיב במערכת מזוהה באופן ייחודי.

טבלה 1: מזהים בפרוטוקול הליגה

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

2.3.1 דוגמאות למזהים

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

2.4 מבנה הודעה כללי – Envelope

כל הודעה בפרוטוקול חייבת לכלול "מעטפת" (Envelope) עם שדות קבועים. הממעטפת 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"  
}
```

שדות חובה במעטפת 2.4.1

טבלה 2: שדות חובה בכל הודעה

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

2.4.2 דרישת אזור זמן – UTC/GMT

חוובה: כל חותמות הזמן בפרוטוקול חייבות להיות באזור זמן UTC/GMT. דרישת זו מבטיחה עקביות בין סוכנים הפעילים ממקומיים גיאוגרפיים שונים.

טבלה 3: פורמטים תקינים ולא תקינים לחותמת זמן

הסבר	תקין?	פורמט
סימנת Z מציין UTC	✓	2025-01-15T10:30:00Z
הפרש +00:00 UTC- לשווה ערך	✓	2025-01-15T10:30:00+00:00
אזור זמן מקומי – אסור	✗	2025-01-15T10:30:00+02:00
לא אזור זמן – אסור	✗	2025-01-15T10:30:00

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

2.4.3 שדות אופציונליים

טבלה 4: שדות אופציונליים לפי הקשר

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

2.4.4 פורמט שדה sender

שדה sender מזהה את שולח הודעה:

- מנהל הליגה – league_manager
- שופט עם מזהה .REF01 – referee:REF01
- שחקן עם מזהה P01 – player: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 לsoccerists אחרים.

קבלת טוקן בתגובה 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 בקשה רישום למנהל הליגה. מנהל הליגה מקצת referee_id ושומר את כתובות השופט.



2.5.2 שלב 2: רישום שחקנים

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



2.5.3 שלב 3: יצרת 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 הודעה ROUND_ANNOUNCEMENT. ההודעה מפרטת את כל המשחקים במהלך. מנהל הליגה מקצת שופט לכל משחק מתוך רשימה של שופטים הרשומים.

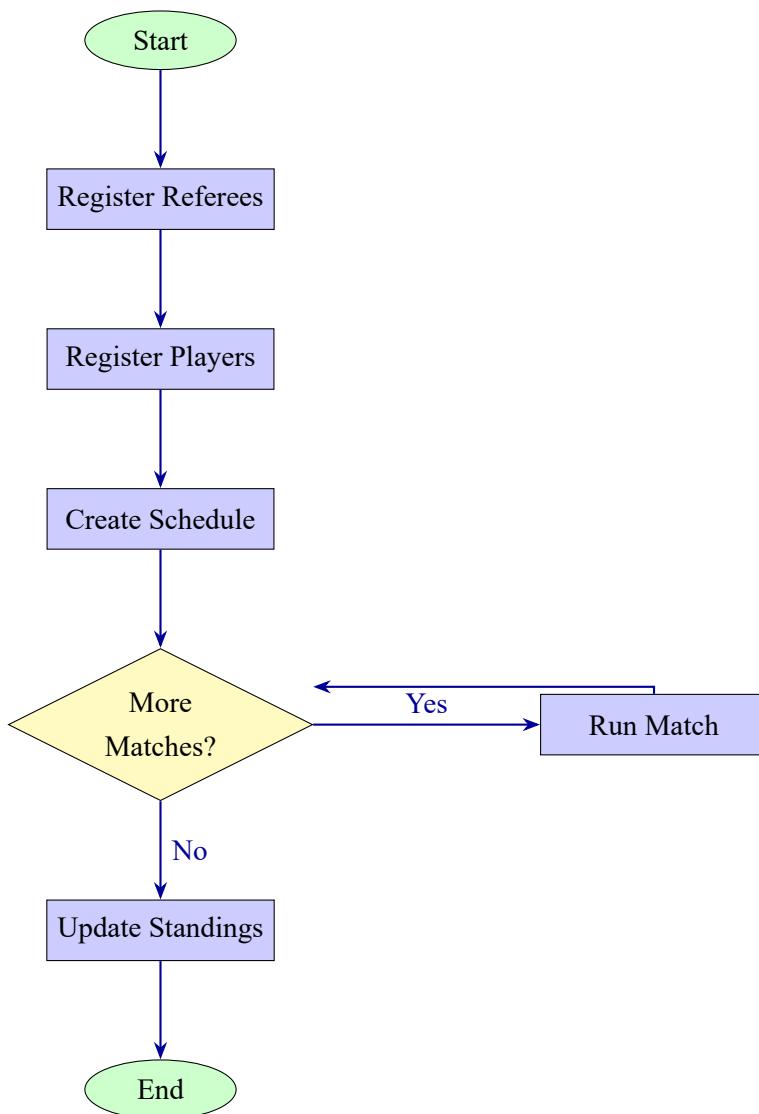
2.5.5 שלב 5: ניהול משחקים

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 הودעה מוגדר זמן תגובה מקסימלי. אם הסוכן לא עונה בזמן, הפעולה נחשבת ככשלון.

טבלה 5: זמני תגובה לפי סוג הודעה

הערות	Timeout	סוג הודעה
רישום שופט לliga	10 sec	REFEREE_REGISTER
רישום שחקן לliga	10 sec	LEAGUE_REGISTER
אישור הגעה למשחק	5 sec	GAME_JOIN_ACK
בחירה זוגי/אי-זוגי	30 sec	CHOOSE_PARITY
קבלת תוצאה משחק	5 sec	GAME_OVER
דיווח תוצאה לliga	10 sec	MATCH_RESULT_REPORT
שאילתת מיידע	10 sec	LEAGUE_QUERY
ברירת מחדל	10 sec	כל השאר

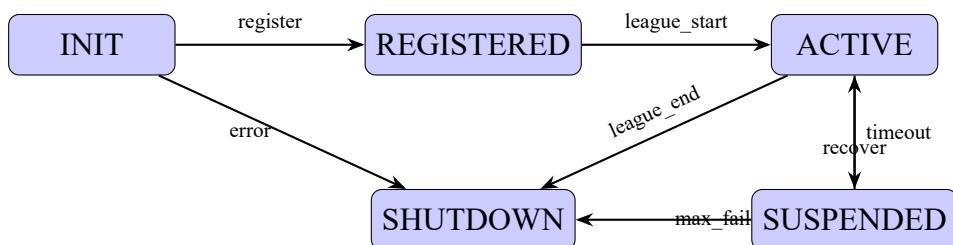
2.8 מחזור חיים של סוכן – Agent Lifecycle

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

2.8.1 מצבים הסוכן

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

2.8.2 דיאגרמת מעברי מצבים



2.9 טיפול בשגיאות

הפרוטוקול מגדיר שני סוגי הודעות שגיאות:

2.9.1 שגיאת ליגה – LEAGUE_ERROR

מנהל הליגה שולח הודעה זו כאשר מתרחשת שגיאה בرمת 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\u00d7ID\u00d7not\u00d7found\u00d7in\u00d7registry",  
    "context": {  
        "player_id": "P99"  
    },  
    "retryable": false  
}
```

2.9.2 שגיאת משחק – GAME_ERROR

השופט שולח הודעה זו כאשר מתרחשת שגיאה במהלך משחק.

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\u00d7not\u00d7received\u00d7within\u00d730\u00d7seconds",  
    "game_state": "COLLECTING_CHOICES",  
    "retryable": true,  
    "retry_count": 1,  
    "max_retries": 3  
}
```

2.9.3 קודי שגיאה נפוצים

טבלה 6: קודי שגיאה עיקריים

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

2.9.4 מדיניות ניסיון חוזר – Retry Policy

שגיאות מסוימות ניתנות לניסיון חוזר:

- מקסימום ניסיונות: 3
 - השהייה בין ניסיונות: 2 שניות
 - **שגיאות ניתנות לניסיון:** (noitcennoc) E009 , (tuoemit) E001 .
- לאחר מיצוי הניסיונות – הפסד טכני (TECHNICAL LOSS).

2.10 תאימות גרסאות

2.10.1 הכרזת גרסה

בעת רישום, כל סוכן מכיריז 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\u2014Alpha",  
        "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 PROTOCOL_VERSION_MISMATCH (MATCH).

2.11 עקרונות חשובים

2.11.1 מקור אמת יחיד

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

2.11.2 תקשורת דרך Orchestrator

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

2.11.3 טיפול בכשלים

אם שחקן לא מגיב:

1. השופט שולח הודעה 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. הבחירה נעשית במקביל, בלי לדעת את בחירת היריב.

4. השופט מגיריל 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

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

3.2 זרימת המשחק בודד

3.2.1 שלב 1: הזמנה למשחק

השופט שולח הזמנה לשני השחקנים. ההזמנה כוללת:

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

3.2.2 **שלב 2: אישור הגעה**

כל שחקן מאשר קבלת הזמנה. האישור כולל חותמת זמן.

3.2.3 **שלב 3: איסוף בחירות**

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

חשוב: השחקנים לא רואים את בחרת היריב.

3.2.4 **שלב 4: הגרלת מספר**

לאחר קבלת שתי הבחירה, השופט מגറיל 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.

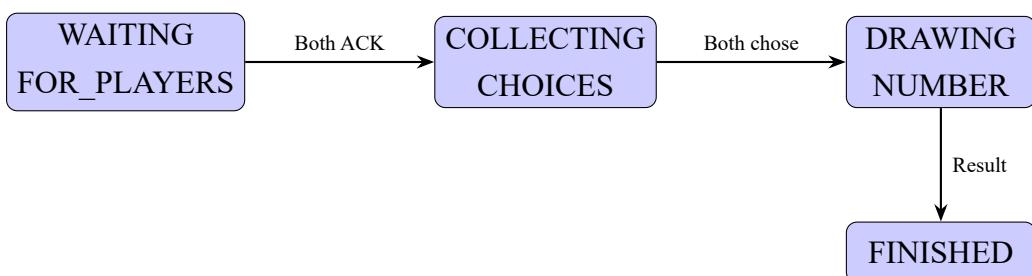
3.2.6 **שלב 6: דיווח תוצאה**

השופט שולח:

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

3.3 **מצבי המשחק**

המשחק עובר בין מצבים מוגדרים:



3.3.1 **מצב WAITING_FOR_PLAYERS**

המשחק מתחילה במצב זה. השופט מכחילה 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: כשתמי הבחירה התקבלו.

3.3.3 מצב DRAWING_NUMBER

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

3.3.4 מצב FINISHED

השחקן הסתiens. התוצאה דוחה.

3.4 שיטת הניקוד

3.4.1 ניקוד למשחק

טבלה 8: טבלת ניקוד

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

3.4.2 דירוג בליגה

הדירוג קבוע 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 משחקי בליגה: $\frac{n(n-1)}{2}$

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

3.5.2 לוח משחקים לדוגמה

טבלה 9: לוח משחקים ל-4 שחקנים

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

3.6 אסטרטגיות לשחקנים

3.6.1 אסטרטגיה אקראית

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

סטרטגיה אקראית

```
import random

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

3.6.2 אסטרטגיה מבוססת היסטוריה

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

3.6.3 אסטרטגיה מונחתית LLM

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

דוגמה ל-prompt

```
prompt = """
You\u201are\u201aplaying\u201aEven/Odd\u201agame.
Choose\u201a"even"\u201aor\u201a"odd".
Previous\u201aresults:\u201aeven\u201awon\u201a3\u201atimes,\u201aodd\u201awon\u201a2\u201atimes.
Your\u201achoice\u201a(one\u201aword\u201aonly):
"""
```

הערה: השימוש ב-LLM מעניין אך לא ישר ביצועים סטטיסטיות. המשחק הוא משחק מזל.

3.7 מודול חוקי המשחק

מודול החוקים הוא רכיב נפרד בשופט. הוא מגדר 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. במקומות 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 הגנרייה המאפשרת הוספת משחקי נוספים.

3.8.1 הפעטה מהלך – GAME_MOVE

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

טבלה 10: הקבלה בין הודעות ספציפיות לגנריות

הודעה גנרית	הודעה ספציפית
GAME_MOVE_CALL	CHOOSE_PARITY_CALL
GAME_MOVE_RESPONSE	CHOOSE_PARITY_RESPONSE

בקשת מהלך גנריית - GAME_MOVE_CALL

```
{
  "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"
}
```

תגובה מהלך גנריית - GAME_MOVE_RESPONSE

```
{
  "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 ההפרשה

1. **הוספת משחקים חדשים** – ללא שינוי בפרוטוקול הבסיסי.
2. **גילוי יכולות** – שחקן יכול לשאול אילו משחקים נתמכים.
3. **ולידציה אחידה** – השופט מודזע שהמהלך חוקי לפי ה-.schema
4. **תאימות קדימה** – סוכנים שונים יכולים להמשיך לעבוד עם הودעות ספציפיות.

הערה: בתרגיל זה משתמשים בהודעות הספציפיות (*_CHOOSE_PARITY). ההפרשה הגנרטית מוצגת לצורך הבנת הארכיטקטורה.

4 מבני הودעות JSON

פרק זה מגדיר את כל הודעות הפרוטוקול. **חשיבות מואוד:** כל הסטודנטים חייבים להשתמש במבנים אלה בדיק. זה אפשר לסטודנטים שלכם לתקשר זה עם זה.

4.1 הודעות רישום שופט לliga

4.1.1 REFeree_REGISTER_REQUEST – בקשה רישום שופט

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

בקשת רישום שופט לliga

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

שדות חובה:

- שם תצוגה של השופט display_name -
- גרסה השופט version -
- רשימת סוגי משחקים שהשופט יודע לשופט game_types -
- כתובת השרת של השופט contact_endpoint -
- מספר משחקים מקסימלי שהשופט יכול לנוהל במקביל max_concurrent_matches -

REFEREE_REGISTER_RESPONSE – 4.1.2

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

תגובה רישום שופט לliga

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

שדות:

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

4.2 הودעות רישום שחקן לliga

4.2.1 בקשה רישום שחקן - LEAGUE_REGISTER_REQUEST

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

בקשת רישום לliga

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

שדות חובה:

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

4.2.2 תגובה רישום – LEAGUE_REGISTER_RESPONSE

- **מאת (From)**: league_manager (מנהל הליגה)
 - **אל (To)**: player (השחקן ששלח את הבקשה)
 - **תגובה צפואה: אין** (הודעת תגובה)
- מנהל הליגה מחזיר תגובה זו.

תגובה רישום לliga

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

שדות:

- status - ."REJECTED" או "ACCEPTED"
- player_id - מזהה שהוקצה לשחקן (רק אם התקבל).
- reason - סיבת דחיה (רק אם נדחה).

4.3 הודעות מחרזור

4.3.1 הכרזת מחרזור - ROUND_ANNOUNCEMENT

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

הכרזת מחרזור

```
{  
    "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)** : השופט המנהל את המשחק (referee)
- **אל (To)** : כל אחד משני השחקנים במשחק (player)
- **תגובה צפוייה:** 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)** : השחקן שקיבל הזמנה (player)
- **אל (To)** : השופט ששלח את הזמנה (referee)
- **תגובה צפוייה:** CHOOSE_PARITY_CALL (לאחר שכל השחקנים מאשרים)

השחקן מאשר קבלת הזמנה.

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 בקשת בחירה - CHOOSE_PARITY_CALL

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

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 תגובה בחירה - CHOOSE_PARITY_RESPONSE

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

תגובה 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" בבדיקה.

4.6 הودעות תוצאה

4.6.1 סיום משחק - GAME_OVER

- **מאט (From)** referee (השופט)
- **אל (To)**: players (שני השחקנים במשחק)
- **תגובה צפוייה**: 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, ועוד).

MATCH_RESULT_REPORT 4.6.2

- **מאת (From)**: referee (השופט שニיהל את המשחק)
- **אל (To)**: league_manager (מנהל הליגה)
- **תגובה צפוייה**: 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"  
            }  
        }  
    }  
}
```

הודעות דירוג 4.7

4.7.1 עדכון דירוג – LEAGUE_STANDINGS_UPDATE

- מאת (From) league_manager : (מנהל הליגה)
- אל (To) : players (כל השחקנים הרשומים)
- תגובה צפואה: 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\u20acAlpha",  
            "played": 2,  
            "wins": 2,  
            "draws": 0,  
            "losses": 0,  
            "points": 6  
        },  
        {  
            "rank": 2,  
            "player_id": "P03",  
            "display_name": "Agent\u20acGamma",  
            "played": 2,  
            "wins": 1,  
            "draws": 1,  
            "losses": 0,  
            "points": 4  
        }  
    ]  
}
```

4.8 הודעות סיום מחזור וליגה

ROUND_COMPLETED - סיום מחזור - 4.8.1

- **מאת (From)** league_manager : (מנהל הליגה)
 - **אל (To)**: players : (כל השחקנים הרשומים)
 - **תגובה צפואה**: 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 ושותפים)
 - תגובה צפואה: 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": "AgentAlpha",  
        "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:

- פרטי האלוף champion
- טבלת הדירוג הסופית final_standings
- סטטיסטיות הליגה total_matches ,total_rounds

4.9 הודעות שאילתא

4.9.1 LEAGUE_QUERY - שאלת ליגה

- מאת (From) player או referee (שחקן או שופט)
 - אל (To): league_manager (מנהל הליגה)
 - תגובה כפואה: LEAGUE_QUERY_RESPONSE
- שחקן או שופט שולח שאלת ליגה לקבלת מידע.

שאילתת 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 שאלות (query_type):

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

4.9.2 **תגובה לשאלתה - LEAGUE_QUERY_RESPONSE**

- **מאת (From)** league_manager : (מנהל הליגה)
- **אל (To)**: referee או player : (השלוח המקורי)
- **תגובה צפואה:** אין (הודעת תגובה)
- מנהל הליגה מחזיר תגובה לשאלתה.

תגובה 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"  
        }  
    }  
}
```

שודות:

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

4.10 **הודעות שגיאה**

שגיאה בرمת הליגה – LEAGUE_ERROR 4.10.1

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

כאשר מתרחשת שגיאה בפעולות ליגה, מנהל הליגה שולח הודעה :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 - שגיאה ברמת המשחק GAME_ERROR

- מאת (From) referee (השופט המנהל את המשחק)
 - אל (To): player (השחקן שגרם לשגיאה או המושפע ממנו)
 - תגובה צפואה: אין (הודעת שגיאה)
- כאשר מתרכחת 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": "TechnicalLossIfMaxRetriesExceeded"  
}
```

שדות:

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

4.11 טבלת סיכום הודעות

4.12 כלליים חשובים

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

מטרה	מקבל	שלוח	סוג הודעה
רישום שופט	ליגה	שופט	REFEREE_REGISTER_REQUEST
אישור רישום שופט	שופט	ליגה	REFEREE_REGISTER_RESPONSE
רישום שחזור	ליגה	שחזור	LEAGUE_REGISTER_REQUEST
אישור רישום שחזור	שחזור	ליגה	LEAGUE_REGISTER_RESPONSE
פרסום מחרוזר	שחקנים	ליגה	ROUND_ANNOUNCEMENT
סיום מחרוזר	שחקנים	ליגה	ROUND_COMPLETED
סיום ליגה	colm	ליגה	LEAGUE_COMPLETED
הזמנה למשחק	שחזור	שופט	GAME_INVITATION
אישור הגעה	שופט	שחזור	GAME_JOIN_ACK
בקשת בחירה	שחזור	שופט	CHOOSE_PARITY_CALL
תגובה בחירה	שופט	שחזור	CHOOSE_PARITY_RESPONSE
סיום משחק	שחקנים	שופט	GAME_OVER
דיווח תוצאה	ליגה	שופט	MATCH_RESULT_REPORT
עדכון דירוג	שחקנים	ליגה	LEAGUE_STANDINGS_UPDATE
שגיאת ליגה	סוכן	ליגה	LEAGUE_ERROR
שגיאת משחק	שחזור	שופט	GAME_ERROR
שאלילתת מידע	ליגה	שחזור/שופט	LEAGUE_QUERY
תגובה שאלילתת	שחזור/שופט	ליגה	LEAGUE_QUERY_RESPONSE

4.12.1 שודות חובה

כל הודעה חייבת לכלול:

- message_type - תמייד.
- match_id - בהודעות משחק.
- player_id - בהודעות שחקנו.

4.12.2 ערכים מותרים

parity_choice - רק או even "odd" (אותיות קטנות).
status - רק WIN DRAW TECHNICAL LOSS.
accept - רק true או false (בוליאני).

4.12.3 פורמט זמן

כל חוותות הזמן בפורמט ISO-8601:

YYYY-MM-DDTHH:MM:SSZ

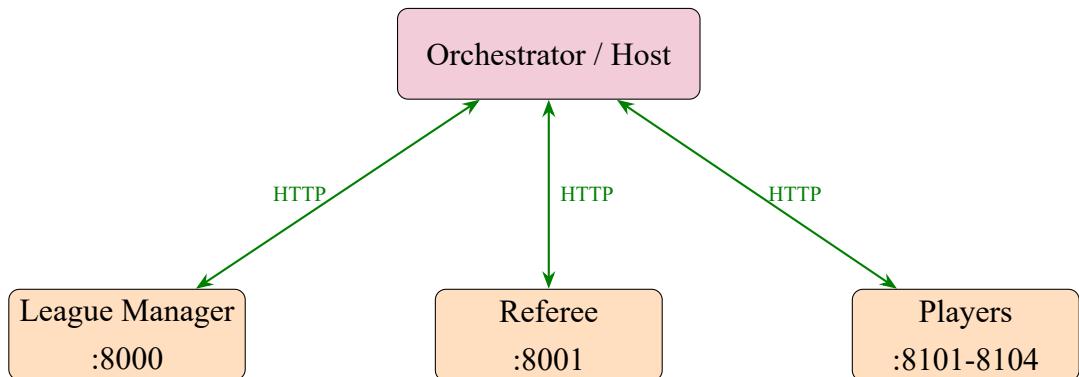
דוגמה: 2025-01-15T10:30:00Z

5 מדריך מימוש

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

5.1 ארכיטקטורה כללית

5.1.1 תרשימים רכיביים



5.1.2 תפקיד ה-Orchestrator

ה-Orchestrator מתאים בין כל הסוכנים. הוא:

- שולח בקשות *HTTP* לכל שירות.
- מקבל תשובות ומעבד אותן.
- מנהל את זרימת הליגה.

מימוש שירות MCP פשוט 5.2

5.2.1 מבנה בסיסי ב-FastAPI

שירות 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)
```

מימוש סוכן שחון 5.3

5.3.1 כלים נדרשים

סוכן שחון חייב למשוך את הכלים הבאים:

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

סוכן שחזור 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 למשוך:

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

5.4.2 רישום שופט לliga

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")
```

5.4.3 לוגיקת קביעת מנצח

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 ניהול ליגה

5.5.1 כלים נדרשים

מנהל הליגה חייב לממש:

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

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)
        }

    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 קרייה 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:

- היסטוריה של משחקים.
- סטטיסטיות אישיות.
- מידע על יריבים.

מחלקה 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 מדוחה למנהל הליגה.

5.9 דפוסי חוסן (Resilience Patterns)

מערכת מבוירת חייבת להתמודד עם כשלים זמניים. ה프וטוקול מגדיר מדיניות ניסיונות חוזרים:

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

לוגיקת ניסיונות חוזרים

```

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

class RetryConfig:
    MAX_ATTEMPTS = 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_ATTEMPTS):
            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_ATTEMPTS - 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:

כircuit 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 מבנה (Structured Logging)

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

טבלה 12: שדות חובה בהודעת לוג

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

Logger מימוש 5.10.1

Logger מבנה

```

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
        }

        if message_type:
            log_entry["message_type"] = message_type

```

שימוש ב-Logger

```
logger = StructuredLogger("player:P01")

# Log received message
logger.info(
    "Received\u00d7game\u00d7invitation",
    message_type="GAME_INVITATION",
    conversation_id="conv-12345",
    data={"match_id": "R1M1", "opponent": "P02"}
)

# Log error
logger.error(
    "Failed\u00d7to\u00d7connect\u00d7to\u00d7referee",
    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\u00d7game\u00d7invitation",
"message_type": "GAME_INVITATION",
"conversation_id": "conv-12345",
"data": {"match_id": "R1M1", "opponent": "P02"}}
```

5.11 אימות וטוקנים (Authentication)

החל מגרסת 2.1.0 של ה프רוטוקול, כל הودעה חייבת לכלול auth_token לאימות. הטוקן מתתקבל בעת הרישום ומשמש לזיהוי הלקוח 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 אימוט

```
class AuthenticatedClient:  
    def __init__(self, credentials: AgentCredentials):  
        self.creds = credentials  
  
    def send_message(self, endpoint: str, message_type: str,  
                    params: dict) -> dict:  
        """SendAuthenticatedMessage."""  
        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()
```

5.11.3 טיפול בשגיאות אימות

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

הrixו 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 לרוֹץ.

2. השופט נרשם למנהל הליגה בעת הפעלה.
3. השחקנים נרשמים למנהל הליגה.
4. רק אז אפשר להתחילה את הליגה.

5.12.2 בדיקת חיבור

בדיקות 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 טיפס למימוש

1. **התחילו פשוט** – ממשו קודם אסטרטגייה אקראית.
2. **בדקו מקומית** – הריצו ליגה עם עצמכם.
3. **שמרו לוגים** – תעדו כל הודעה.
4. **טפו בשגיאות** – השתמשו ב-try/except.
5. **עקבו אחר הпрוטוקול** – השתמשו במבנה JSON בדיק.