

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 | ארQUITקטורת 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 גנריית | |
| 32 | 3.8.3 רישום סוגים משחקים – Game Registry | |
| 32 | 3.8.4 יתרונות ההפשטה | |
| 33 | 4 מבני הודעות JSON | |
| 33 | 33.1 הודעות רישום שופט לליגה | 4.1 |
| 33 | 33.1.1 REFEREE_REGISTER_REQUEST – בקשה רישום שופט | 4.1.1 |
| 34 | 33.1.2 REFEREE_REGISTER_RESPONSE – תגובה רישום שופט | 4.1.2 |
| 35 | 33.2 הודעות רישום שחקן לליגה | 4.2 |
| 35 | 33.2.1 LEAGUE_REGISTER_REQUEST – בקשה רישום שחקן | 4.2.1 |
| 36 | 33.2.2 LEAGUE_REGISTER_RESPONSE – תגובה רישום | 4.2.2 |
| 37 | 33.3 הודעות מחזר | 4.3 |
| 37 | 33.3.1 ROUND_ANNOUNCEMENT – הכרזת מחזר | 4.3.1 |
| 37 | 33.4 הודעות משחק | 4.4 |
| 38 | 33.4.1 GAME_INVITATION – הזמנה למשחק | 4.4.1 |
| 38 | 33.4.2 GAME_JOIN_ACK – אישור הגעה | 4.4.2 |
| 38 | 33.5 הודעות בחירה במשחק זוגי/אי-זוגי | 4.5 |
| 39 | 33.5.1 CHOOSE_PARITY_CALL – בקשה בחירה | 4.5.1 |
| 39 | 33.5.2 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 שרת MCP פשוט | 5.2 |
| 52 | מבנה בסיסי ב-FastAPI | 5.2.1 |
| 52 | מימוש סוכן שחkon | 5.3 |
| 52 | כלים נדרשים | 5.3.1 |
| 53 | מימוש לדוגמה | 5.3.2 |
| 54 | מימוש שופט | 5.4 |
| 54 | כלים נדרשים | 5.4.1 |
| 54 | רישום שופט לliga | 5.4.2 |
| 55 | לוגיקת קביעת מנצח | 5.4.3 |
| 55 | מימוש מנהל liga | 5.5 |
| 55 | כלים נדרשים | 5.5.1 |

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

| | | |
|----|---|-------|
| 70 | שלב 1: פיתוח מקומי | 6.4.1 |
| 70 | שלב 2: ליגה פרטית | 6.4.2 |
| 71 | שלב 3: בדיקת תאימות עם סטודנטים אחרים | 6.4.3 |
| 71 | מבט לעתיד: ליגת כיתה | 6.4.4 |
| 71 | הגשה | 6.5 |
| 71 | קבצים להגשת | 6.5.1 |
| 71 | פורמט הגשה | 6.5.2 |
| 71 | דגשים כלליים לבדיקת העבודה | 6.6 |
| 72 | שאלות נפוצות | 6.7 |
| 72 | האם אפשר להשתמש בספריות חיצונית? | 6.7.1 |
| 72 | האם חיברים 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.json | 9.5.1 |
| 110 | לוג סוכן – logs/agents/<agent_id>.log.json | 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 | מבנה סוכן טיפוסי | 11.4.1 |
| 127 | תקיית התיעוד – /doc | 11.5 |
| | 11.6. דיאגרמת ארכיטקטורה | 129 |
| 129 | זרימת נתונים | 11.7 |
| 129 | קריאה וכתיבה | 11.7.1 |

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

133

12 **מקורות**

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 מטרת 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.

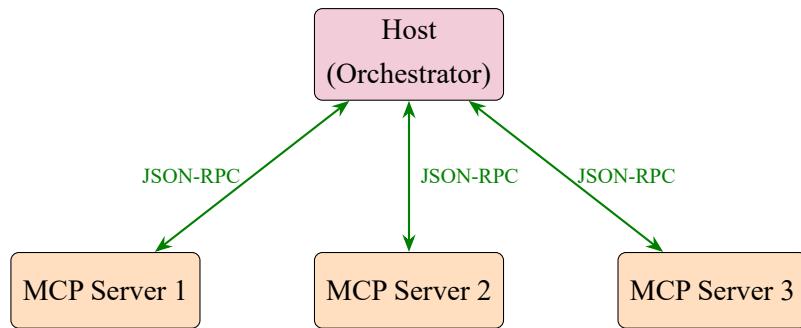
1.2.1 עקרונות הפרוטוקול

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 שירותי המארח שלוח 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` `localhost:8000/mcp`.

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 לסוכני AI. המערכת תכלול שלושה סוגי סוכנים:

1. **מנהל ליגה** (League Manager) – מנהל 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 ומקצת מהם 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 מנץח.
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 |
| חוותמת 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 | תיקין? | 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 רישום (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.

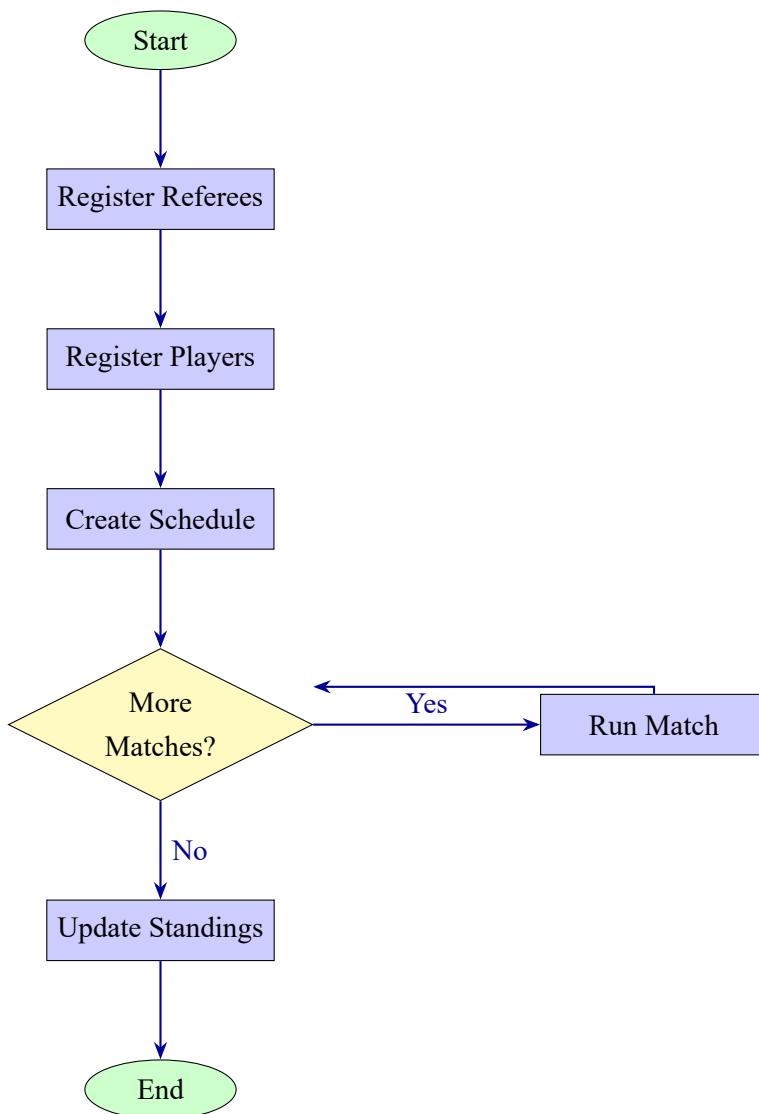
2.5.5 שלב 5: ניהול משחקים

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 הودעה

| 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 |

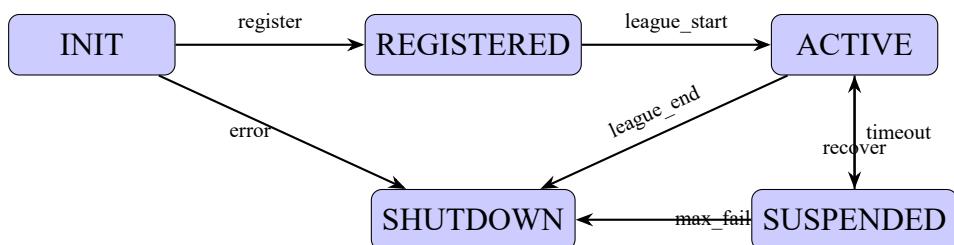
2.8 מחזור חיים של סוכן – Agent Lifecycle

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

2.8.1 מצבים הסוכן

- INIT – הסוכן הופעל אך טרם נרשם.
- .auth_token - REGISTERED – 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 שניתנות לניסיון:** (noitcennoc) E009 , (tuoemit) E001 (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 מסתמכים 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 בחריות**

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.

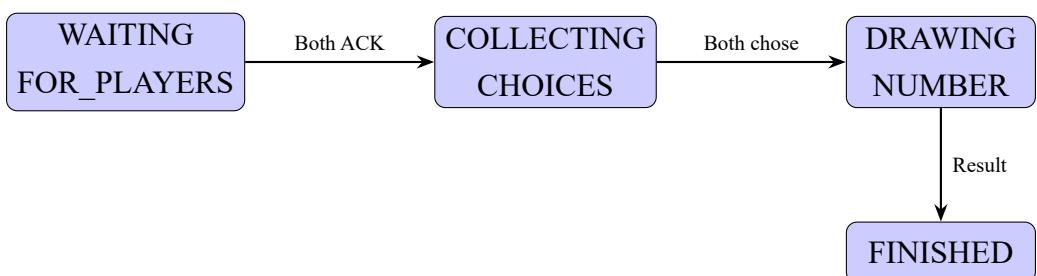
3.2.6 **שלב 6: דיווח תוצאה**

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 מחייב 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 לאוזות דפוסים בהגרלות.
הערה: מכיוון שההגרלה אקראית, אסטרטגיה זו לא תשפר תוצאות לטווח ארוך.

3.6.3 אסטרטגיה מונחת 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 הגנריית 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 הודעות 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 - 4.8.1 סיום מחזור

- **מאט (From)** league_manager : (מנהל הליגה)
 - **אל (To)**: players (כל השחקנים הרשומים)
 - **תגובה צפוייה**: 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 ליגת - LEAGUE_COMPLETED

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

הודעת סיום ליגת

```
{  
    "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:

- פרט 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 (שחקן שגרם לשגיאה או המושפע ממנו)
 - תגובה צפואה: אין (הודעת שגיאה)
- כאשר מתרכשת 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.
- בהודעות משחק match_id -
- בהודעות שחקן player_id -

4.12.2 ערכים מותרים

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

4.12.3 פורמט זמן

כל חוותות הזמן בפורמט ISO-8601:

YYYY-MM-DDTHH:MM:SSZ

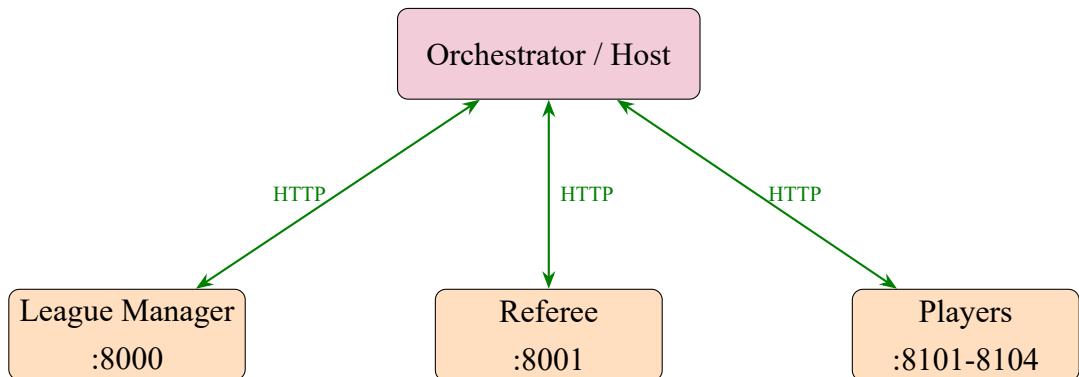
דוגמה: 2025-01-15T10:30:00Z

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 משחק 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 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:

- 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.

6 דרישות תרגיל הבית

6.1 מטרת התרגיל

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

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

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

תרגיל זה מבוסס על הספר:

סוכני בינה מלאכותית עם פרוטוקול הקשר המודול

AI Agents with Model Context Protocol

מאת ד"ר יורם סגל

December 9, 2025

לכן מאד מומלץ לקרוא וללמוד את הנושא לעומק.

6.2 משימות חובה

6.2.1 משימה 1: מימוש סוכן שחקן

ממשו שירות MCP שמאזין על פורט ב-localhost. השירות חייב לתמוך בכלים הבאים:

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

6.2.2 משימה 2: רישום לliga

הסוכן חייב לשЛОוח בקשה רישום למנהל הליגה. הבקשה תכלול:

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

6.2.3 משימה 3: בדיקה עצמית

לפני ההגשה, בדקו את הסוכן שלכם:

1. הריצו ליגה מקומית עם 4 שחקנים.

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

6.3 דרישות טכניות

6.3.1 שפת תכנות

אתם יכולים לבחור כל שפה שתרצו. העיקר שהסוכן:

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

6.3.2 זמני תגובה

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

6.3.3 יציבות

הסוכן חייב:

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

6.4 תהליך העבודה

6.4.1 שלב 1: פיתוח מקומי

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

6.4.2 שלב 2: ליגה פרטיט

1. הריצו ליגה מוקומית עם 4 עותקים של הסוכן.
2. בדקו שככל התקשרות עובדת.
3. שפרו את האסטרטגייה (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 לפרטוקול.

6.4.4 מבט לעתיד: ליגת כיתה

הערה חשובה

בעתיד, ייתכן שתידרשו:

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

נושא זה טרם נסגר וייתכנו שינויים. עלייכם להיערך לכך ולבנות את הסוכן בצורה גמישה שתאפשר הרחבת 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 והIMPLEMENTATION.
 - 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 לריפזיטורי כשהוא ציבורי. ויש להגיש בנהל ההגשה הרגיל כפי שהוצעו התרגילים הקודמים.

6.6 דגשים כלליים לבדיקת העבודה

מעבר לדרישות הרגילות, הクリיטריונים הבאים ייבדקו:

טבלה 13: קriterיונים לבדיקה

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

6.7 שאלות נפוצות

6.7.1 האם אפשר להשתמש בספריות חיצונית?

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

6.7.2 האם חיבבים להשתמש ב-Python?

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

6.7.3 מה קורה אם הסוכן שלי קורס?

הסוכן יספג הפסד טכני במשחק הנוחתי. אם הוא לא חוזר לפועלה – הוא יוצא מהליגה.

6.7.4 האם אפשר לעדכו 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 מחזיר. תוכלו לראות את המיקום של הסוכן שלכם.

6.8 סיכום

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

בצלחה!

מידע נוסף:

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

7 למידת MCP דרך תרגיל הליגה

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

7.1 השחקן כסוכן AI

7.1.1 האם סוכן השחקן הוא סוכן AI?

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

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

7.1.2 ארבעת המאפיינים של סוכן AI

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

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

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

3. **פעולה** – הסוכן משפייע על הסביבה. השחקן מבצע פעולות על ידי שליחת בחירות (GAME_INVITATION), בקשות לבחירת זוגיות (GAME_JOIN_CALL), ותוצאות משחקים (GAME_RESPONSE) (CHOOSE_PARITY_RESPONSE).

4. **תכליתיות** – יש לו מטרה מוגדרת. מטרתו היא לשחק, לנצח משחקים 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 מגדים "חשיבה" או "הסקת מסקנות" לפני ביצוע הפעולה.

7.2 השחקן בארכיטקטורת MCP

7.2.1 שרת או ל��וח?

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

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

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

7.2.2 היחסים מול השופט ומנהל הליגה

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

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

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

7.3 השופט ומנהל הליגה כסוכני AI

7.3.1 סוכנים בדרجة גבוהה

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

7.3.2 שירותי MCP שפועלים גם בכלוקות

שני הסוכנים הללו מוגדרים כשירות MCP:

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

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

טבלה 14: מאפייני סוכן AI עבור השופט ומנהל הליגה

| מאפיין | מנהל ליגת | שופט |
|------------|--|---|
| תכליתיות | ניהול הליגה כולה, רישום שופטים וশחקנים, ייצירת לוח משחקים, חישוב דירוג | רישום למנהל הליגה, ניהול משחק בודד, אימות חוקיות מהלכים, קביעת מנצח |
| אוטונומיות | פועל באופן עצמאי לרישום שופטים ולקביעת סבבי משחק | נרשם באופן עצמאי ליגת ומנהל את שלבי המשחק |
| תפיסה | קולט בקשות רישום משופטים וশחקנים, תוצאות מהשופטים | קולט אישורי הגעה, בחירות זוגיות/אי-זוגיות מהשחקנים |
| פעולה | אשר רישום שופטים וশחקנים, שלוח הכרזות מחזור, מעدقן טבלאות דירוג | שלוח בקשת רישום ליגת, שלוח הזמנות מושך, בקשות בחירה, מדוח תוצאות |

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

7.4 היפוך התפקידים: תובנה מרכזית

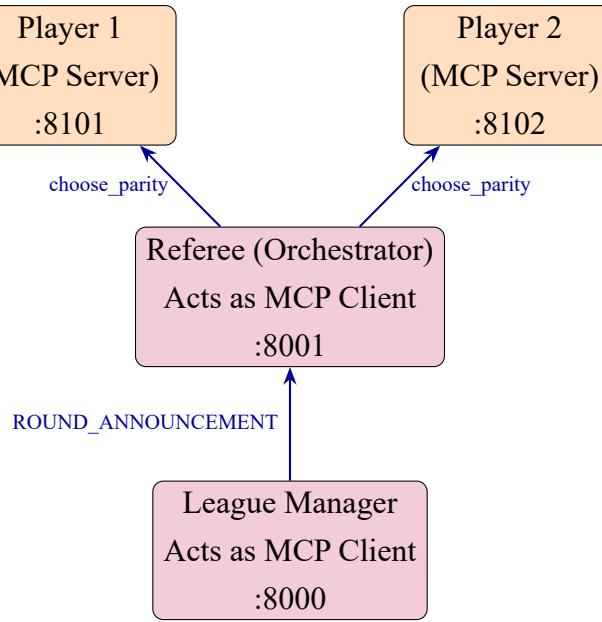
7.4.1 הפרדיגמה המסורתית

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

7.4.2 היפוך התפקידים בליגה

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

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



7.5 עקרון הפרדת השכבות

7.5.1 שלוש שכבות נפרדות

פרוטוקול MCP מאפשר הפרדה ברורה בין התפקידים:

1. **שכבה הליגה** (מנוהלת על ידי מנהל הליגה) – גיוס שחקנים, לוח משחקים - (Round-Robin, וטבלת דירוג).
2. **שכבה השיפוט** (מנוהלת על ידי השופט) – ניהול משחק בודד ואיומות מהלכים.
3. **שכבה חוקי המשחק** (מנוהלת על ידי מודול נפרד) – הלוגיקה הספציפית למשחק זוגי/אי-זוגני.

7.5.2 היתרון של ההפרדה

השחקן, בכך שהוא חושף ממשק MCP סטנדרטי (JSON-RPC 2.0 על גבי HTTP), מאפשר ליגא להישאר אגנוזטיבית לשפת הפיתוח או לאסטרטגייה הפנימית שלו.

זהו פתרון לביעית הrogrammatical שבה לכל סוכן ולכל מודול נדרשה בעבר אינטגרציה ייחודית. פרוטוקול MCP פותר זאת על ידי יצירת ממשק אוניברסלי [2].

כאשר השחקן מקבל בקשה כמו CHOOSE_PARITY_CALL, הנתונים מגיעים במבנה JSON קבוע. השחקן מגיב 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 השירות

7.6.1 הדילמה

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

7.6.2 הפתרון: הפרדת תפקידיים

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

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

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

כאשר השירות מקבל בקשה choose_parity:

1. שכבת ה-MCP (השרת) קולטת את הבקשת.
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 השירות, והוא אינו מפרק את מודל השירות-לקוח. הרעיון המרכזי ב-MCP הוא להבטיח שגם כאשר ה"מוח" נמצא בתוך השירות, התקשורות החיצונית תישאר סטנדרטית באמצעות JSON-RPC.

7.6.3 אנלוגיה: תחנת שירותים לקוחות

ניתן לדמיין את הארכיטקטורה כתחנת שירותים לקוחות:

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

7.7 תפקיד האורקסטרטור

7.7.1 מנהל הליגה – הארכיטקט

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

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

7.7.2 השופט – המימוש הדינמי

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

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

7.8 מה התרגיל מלמד

7.8.1 עקרונות יסוד של סוכני AI

התרגיל מלמד את ארבעת המאפיינים של סוכן AI באופן מעשי:

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

7.8.2 עקרונות יסוד של MCP

התרגיל מלמד את עקרונות הליבה של פרוטוקול MCP:

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

7.8.3 חווית הלמידה

בסיום התרגיל, הסטודנט יבין:

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

7.9 סיכום

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

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

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

8 הרצת מערכת הליגה

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

8.1 תצורת המערכת

8.1.1 פורטימן וטרמינלים

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

טבלה 15: הקצאת פורטימן 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 המחזוריים.
- **השופטים** – Orchestratorים מקומיים למשחק בודד. כל שופט הוא מקור האמת למצב המשחק שלו.

8.2 סדר הפעלה

8.2.1 עקרון סדר הפעלה

סדר הפעלה קריטי לתפקוד תקין של המערכת:

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

8.2.2 טרמינל 1 – מנהל הליגה

הפעלת מנהל הליגה

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

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

8.2.3 טרמינלים 2-3 – שופטים

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-ERE_EEE_REGISTER_REQUEST למנהל הליגה.

הפעלת שחקנים

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

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

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

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

כל שחקן שולח LEAGUE_REGISTER_REQUEST למנהל הליגה.

8.3 שלב 1: רישום שופטים

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

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

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

REFEREE_REGISTER_RESPONSE – תגובה רישום שופט

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

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

8.4 שלב 2: רישום שחקנים

לאחר שהשופטים נרשמו, כל שחקן שולח בקשה רישום.

8.4.1 בקשה רישום שחקן

LEAGUE_REGISTER_REQUEST – בקשה רישום שחקן

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

league_register_response - תגובה רישום שחקן

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

באופן דומה:

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

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

8.5 שלב 3: יצרת לוח משחקים

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

8.5.1 לוח משחקים לארבעה שחקנים

טבלה 16: לוח משחקים Round-Robin לארבעה שחקנים

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

8.6 שלב 4: הכרזה על מחזור

מנהל הליגה שולח לכל השחקנים הודעה .ROUND_ANNOUNCEMENT

הכרזה על מחזור – ROUND_ANNOUNCEMENT

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

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

8.7 שלב 5: ניהול משחק בודד

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

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

הזמנה ל-P01:

P01-ל GAME_INVITATION – הזמנה למשחק

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

הזמנה ל-P02:

P02-ל GAME_INVITATION - הזמנה למשחק

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

8.7.2 שלב 2: אישורי הגעה

כל שחקן מוחזיר GAME_JOIN_ACK תוך 5 שניות.

:P01-מ אישור

P01 GAME_JOIN_ACK – אישור הגעה

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

:P02 אישור מ-

P02 GAME_JOIN_ACK – אישור הגעה

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

כasher haShofet kibl shni ACK chiyobim beZman hamotter, hoa meuber at matzav haMeshekh .COLLECTING_CHOICES-l

8.7.3 שלב 3: איסוף בחירות

השופט שולח שולח CHOOSE_PARITY_CALL לכל שחקן.

בקשת בחירה ל-P01:

P01-ל CHOOSE_PARITY_CALL - בקשת בחירה

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

תגובה P01 (בחר "even"):

P01- CHOOSE_PARITY_RESPONSE – תגובה בחירה

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

תגובה P02 (בחירה "odd"):

P02- CHOOSE_PARITY_RESPONSE – תגובה בחירה

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

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

8.7.4 שלב 5.4: הגרלת מספר וקבעת מנץח

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

drawn_number = 8 -

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

8.7.5 שלב 5.5: הודעת סיום לשחקנים

השופט שולח GAME_OVER לשני השחקנים:

סיום משחק – GAME_OVER

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

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

8.7.6 שלב 6: דיווח למנהל הליגה

השופט מדוח דיווח למנהל הליגה: MATCH_RESULT_REPORT

דיווח תוצאה – MATCH_RESULT_REPORT

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

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

8.8 שלב 6: סיום מחזור ועקבון דירוג

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

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

עדכון דירוג – LEAGUE_STANDINGS_UPDATE

```
{  
    "jsonrpc": "2.0",  
    "method": "update_standings",  
    "params": {  
        "protocol": "league.v2",  
        "message_type": "LEAGUE_STANDINGS_UPDATE",  
        "sender": "league_manager",  
        "timestamp": "2025-01-15T10:20:00Z",  
        "conversation_id": "conv-round-1-standings",  
        "league_id": "league_2025_even_odd",  
        "round_id": 1,  
        "standings": [  
            {  
                "rank": 1,  
                "player_id": "P01",  
                "display_name": "Agent\u20acAlpha",  
                "played": 1,  
                "wins": 1,  
                "draws": 0,  
                "losses": 0,  
                "points": 3  
            },  
            {  
                "rank": 2,  
                "player_id": "P03",  
                "display_name": "Agent\u20acGamma",  
                "played": 1,  
                "wins": 0,  
                "draws": 1,  
                "losses": 0,  
                "points": 1  
            },  
            {  
                "rank": 3,  
                "player_id": "P04",  
                "display_name": "Agent\u20acDelta",  
                "played": 1,  
                "wins": 0,  
                "draws": 1,  
                "losses": 0,  
                "points": 1  
            },  
            {  
                "rank": 4,  
                "player_id": "P02",  
                "display_name": "Agent\u20acBeta",  
                "played": 1,  
                "wins": 0,  
                "draws": 0,  
                "losses": 1  
            }  
        ]  
    }  
}
```

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

סיום מഴור - ROUND_COMPLETED

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

8.9 שלב 7 : סיום הליגה

לאחר סיום כל המחזוריים, מנהל הליגה שולח הודעה LEAGUE_COMPLETED

סיום ליגת – LEAGUE_COMPLETED

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

8.10 טיפול בשגיאות

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

LEAGUE_ERROR – שגיאות אימות

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

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 כלים שאילתות זמינים

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

8.11.1 שאלת דירוג מנהל הליגה

שחקן שורצוה לוודא את דירוגו קורא למנהל הליגה:

שאילתת דירוג – LEAGUE_QUERY

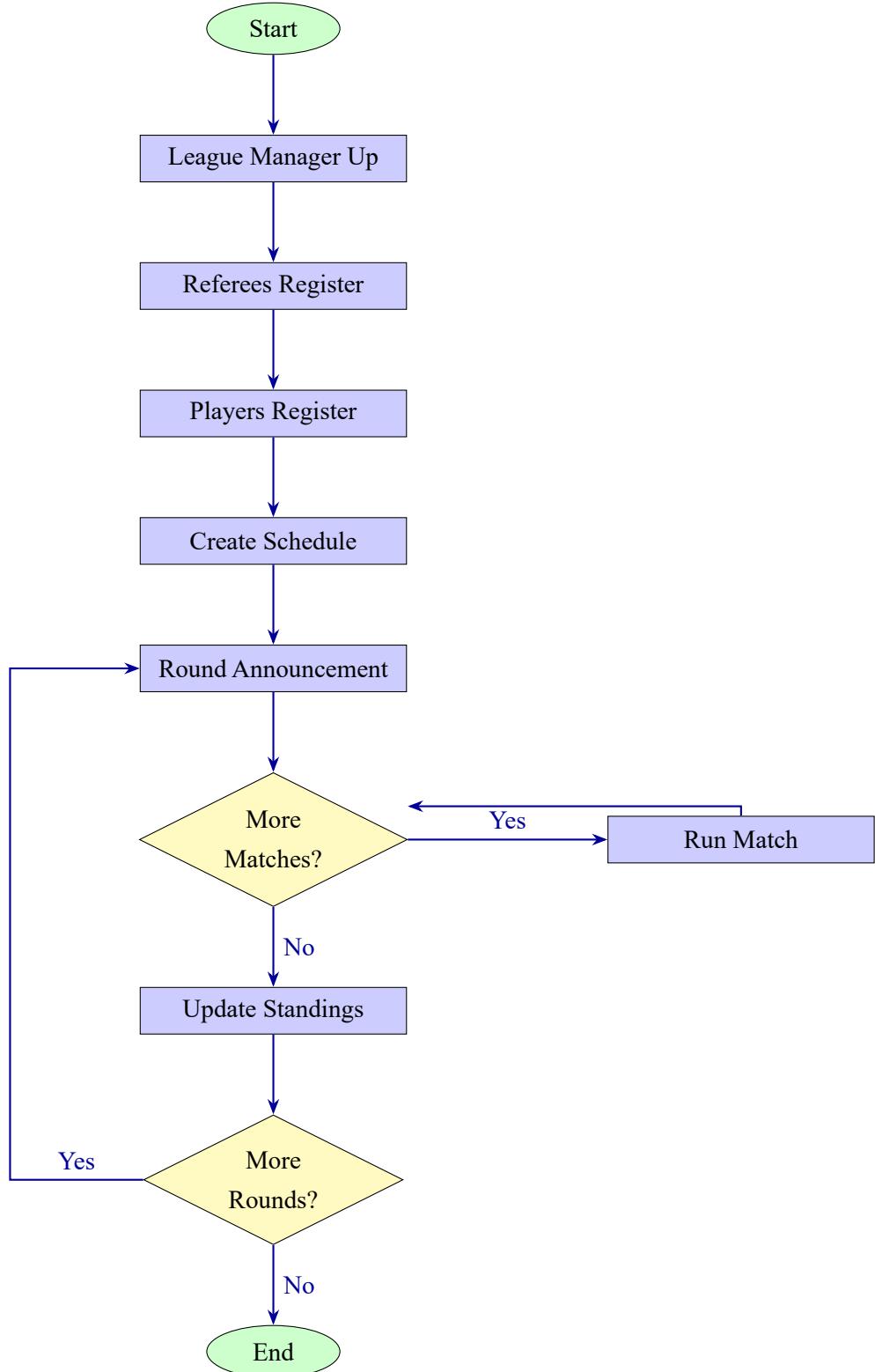
```
{  
    "jsonrpc": "2.0",  
    "method": "league_query",  
    "params": {  
        "protocol": "league.v2",  
        "message_type": "LEAGUE_QUERY",  
        "sender": "player:P01",  
        "timestamp": "2025-01-15T10:25:00Z",  
        "conversation_id": "conv-query-standings-001",  
   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

מנהל הליגה מחזיר מחולל שכלל אובייקט result בפורמט זהה 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 דיבוב).
- **כלי בשחקן:** get_player_state – נותן את ההיסטוריה המשחקים.

8.12 דיאגרמת זרימה מלאה



8.13 טבלת תפקידי הסוכנים

טבלה 17: תפקידי הסוכנים במערכת

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

8.14 סיכום

נספח זה הציג את:

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

כל התקשרות מתבצעת באמצעות JSON-RPC 2.0 over HTTP. כל ההודעות כוללות מוטף (Envelope) אחד 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

כasher anu bonyim chibra shel socnims ottonomiyim - shaknim, shoftim v'manhali ligah - anu lema'aseh yozrim terbotot digitalit chadsha. cmu b'khol chibra anoushit, g'm ca'an n'drashim shelosha y'sodot kritiyim:

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

Nespa' zeh matayr at "basis ha'ntoniim ul kovchi JSON" - architekturat shelosh shabotot she'mafsharat la'meracat l'zmo'ah l'kena' midah shel alpi socnims v'ligot.

9.2 ארכיטקטורת שלוש השכבות



9.2.1 עקרונות מנהים

כל kovch b'meracat umid be'ukronot ha'ba'im:

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

9.3 שכבת הקונפיגורציה – config/

Shcbeh zo mchila at ha'kode' ha'geneti' shel meracat - hagdrot statiyot shnkravot be'uliyot ha'socnims.

9.3.1 קובץ מערכת גלובלי - config/system.json

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

דוגמה: מבנה system.json (קטע)

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

9.3.2 רישום סוכנים - config/agents/agents_config.json

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

9.3.3 קונFIGורציית ליגה - config/leagues/<league_id>.json

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

דוגמה: קונFIGורציה ליגה (קטע)

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

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

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

9.3.5 ברירות מחדל לסוכנים - config/defaults/

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

9.4 שכבת נתונים הריצה - data/

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

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

- מטרה: מצב הדירוג העדכני של הליגה.

- מערכן: מנהל הליגה (אחרי MATCH_RESULT_REPORT).

- מיקום: SHARED/data/leagues/league_2025_even_odd/standings.json

דוגמה: מבנה טבלת דירוג

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

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

- מטרה: תיעוד כל המחזוריים שהתקיימו.

- מערכן: מנהל הליגה (אחרי ROUND_COMPLETED).

- מיקום: SHARED/data/leagues/league_2025_even_odd/rounds.json

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

- מטרה: תיעוד מלא של משחק בודד.

- מערכן: השופט שניהל את המשחק.

- מיקום: SHARED/data/matches/league_2025_even_odd/R1M1.json

קובץ זה הוא ה"תעודת זהות" של המשחק ומכיל:

- מצב המשחק וזמן.

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

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

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

דוגמה: היסטוריה שחון

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

9.5 שכבת הלוגים – logs/

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

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

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

דוגמה: רשומה 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 שלח 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 בסעיפים זה זמינים בתיקייה המשותפת:

L07 / SHARED /

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

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

טבלה 18: סיכום קובצי הקונפיגורציה והנתונים

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

9.8 סיכום

ארכיטקטורת שלוש השכבות שהציגו – קונפיגורציה, נתוני ריצה, ולוגים – מספקת את התשתיות הנדרשת לבניית מערכת סוכנים בקנה מידה גדול. כמו בחברה אנושית, גם כאן:

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

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

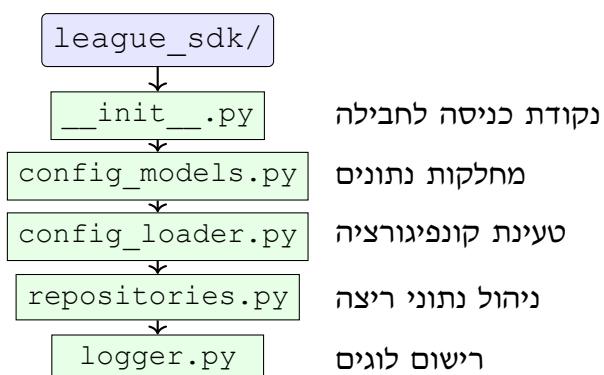
10. ערכות כלים בפייתון

10.1 מבוא: מהקונפיגורציה לךוד

בנספח ב הצנו את ארכיטקטורת הנתונים מבוססת קובץ JSON – שלוש שכבות של קונפיגורציה, נתונים ריצה, ולוגים. אך כיצד סוכן בודד ניגש נתונים אלו בפועל? נספח זה מציג את league_sdk – ספרייה שմחברת בין קובצי ה-JSON לבין האובייקטים שבהם משתמשים הסוכנים. הספרייה מימושת שני דפוסי עיצוב מרכזיים:

1. מודלים טיפוסיים (typed models) שימושים את מבנה ה-JSON.
2. שכבת הפשטה לגישה נתונים. Repository Pattern

10.2 מבנה הספרייה



10.3 מודלים טיפוסיים – config_models.py

10.3.1 הגישה: Dataclasses

Python 3.7+ מספקת את 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 קונפיגורציה מערכת

הקובץ מגדר את כל המודלים התואימים ל-system.json

מודלי קונפיגורציה גלובלית

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

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

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

10.3.3 מודלי סוכנים

כל סוג סוכן מקבל מחלוקת קונפיגורציה משלהו:

מודלי קונפיגורציית סוכנים

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

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

10.3.4 מודלי ליגה

הגדירות ספציפיות לliga כוללות תזמון, ניקוד, ומשתתפים:

מודלי קונפיגורציית ליגה

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

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

10.4 טווען קונפיגורציה – ConfigLoader

10.4.1 העיקרונות: טעינה עצלה עם מטמון

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

מבנה ConfigLoader

```
class ConfigLoader:  
    def __init__(self, root: Path = CONFIG_ROOT):  
        self.root = root  
        self._system = None          # lazy cache  
        self._agents = None          # lazy cache  
        self._leagues = {}           # league_id -> LeagueConfig  
  
    def load_system(self) -> SystemConfig:  
        """Load global system configuration."""  
        if self._system:  
            return self._system  
        path = self.root / "system.json"  
        data = json.loads(path.read_text(encoding="utf-8"))  
        self._system = SystemConfig(...)  
        return self._system
```

10.4.2 שיטות הטעינה

מספקת ממשק אחד לטעינת כל סוגי הקונפיגורציה ConfigLoader

טבלה 19: שיטות הטעינה של ConfigLoader

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

10.4.3 שיטות עזר

בנוסף לטעינה ישירה, המחלקה מספקת שיטות נוחות לחיפוש:

שיטות עזר

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

10.5 מאגרי נתונים – Repositories

10.5.1 דפוס המאגר (Repository Pattern)

בעוד Sh-ConfigLoader מטפל בكونFIGורציה סטטית, שכבת המאגרים מטפלת בתונינים דינמיים. כל מאגר אחראי על קרייה, עדכון, ושמירה של סוג נתונים ספציפי.

10.5.2 מאגר טבלת דירוג – StandingsRepository

מאגר טבלת דירוג

```
class StandingsRepository:  
    def __init__(self, league_id: str, data_root: Path = DATA_ROOT):  
        self.league_id = league_id  
        self.path = data_root / "leagues" / league_id / "standings.json"  
        self.path.parent.mkdir(parents=True, exist_ok=True)  
  
    def load(self) -> Dict:  
        """LoadStandingsFromJSONFile."""  
        if not self.path.exists():  
            return {"schema_version": "1.0.0", "standings": []}  
        return json.loads(self.path.read_text(encoding="utf-8"))  
  
    def save(self, standings: Dict) -> None:  
        """SaveStandingsToJsonFile."""  
        standings["last_updated"] = datetime.utcnow().isoformat() + "Z"  
        self.path.write_text(json.dumps(standings, indent=2))  
  
    def update_player(self, player_id: str, result: str, points: int):  
        """UpdateAPlayer'sStandingsAfterAMatch."""  
        standings = self.load()  
        # ... update logic  
        self.save(standings)
```

10.5.3 מאגרים נוספים

הספרייה כוללת מאגרים נוספים לניהול נתונים ריכוז:

טבלה 20: מאגרי נתונים זמינים

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

10.6 רישום לוגים – JsonLogger

10.6.1 פורמט JSON Lines

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

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

10.6.2 מחלקה הלוגר

מחלקה JsonLogger

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

10.6.3 שיטות נוחות

הלוגר מספק שיטות לרמות לוג שונות ולאירועים נפוצים:

שיטות נוחות ללוגר

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

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

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

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

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

10.7 שימוש בסוכנים

10.7.1 דוגמה: מנהל ליגת

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
```

רישום שגיאת TIMEOUT

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

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

10.8 סיכום

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

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

שימוש בספרייה זו מבטיח:

1. **עקביות** – כל הסוכנים משתמשים באותם מודלים 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 זינה בתיקייה:

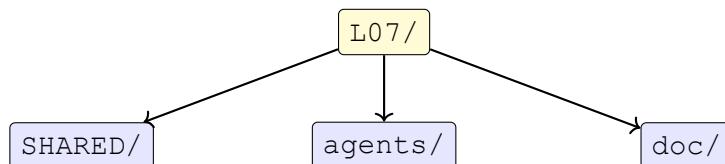
L07/SHARED/league_sdk/

11 מבנה הפרויקט

11.1 מבוא: מפת הדרכים

לאחר שהכרנו את ה프וטוקול, הودעות ה-JSON, ארכיטקטורת הנתונים, וספריית ה-Python – הגיע הזמן לראות את התמונה המלאה. נספח זה מציג את מבנה התיקיות והקבצים המומלץ לפרויקט הליגה, כך שכל סטודנט יוכל להתחיל לעבוד עם בסיס מסודר 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 בתיקייה נפרד |
| doc/ | תיעוד הפרויקט – מסמכים ומפרטים |

11.3 תיקית המשאבים המשותפים – /SHARED/

תיקיה זו מכילה את כל המשאבים המשותפים לכל הסוכנים במערכת.

SHARED/

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

```

    └── player.json          # Default player settings

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

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

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

```

11.4 תקיות הסוכנים – agents/

כל סוכן חי בתיקייה נפרדת עם מבנה אחיד:

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

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

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

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

11.4.1 מבנה סוכן טיפוסי

כל סוכן מכיל את הקבצים הבאים:

11.5 תקיות התיעוד – doc/

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

טבלה 22: קבצי סוכן טיפוסי

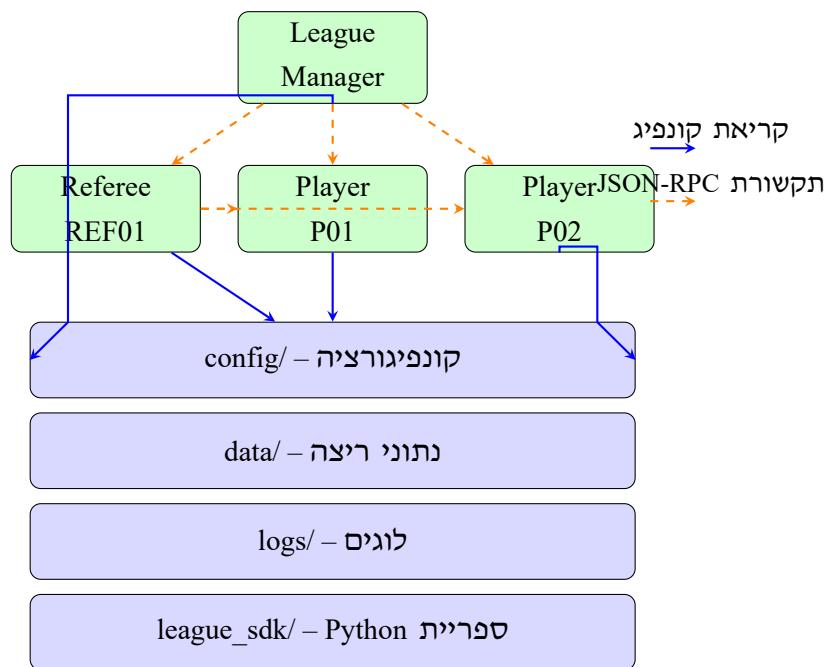
| קובץ | תפקיד |
|------------------|---|
| main.py | נקודות כניסה - אתחול השירות וטעינה קונפיגורציה |
| handlers.py | טיפול בהודעות כניסה לפי סוג |
| strategy.py | לוגיקת קבלת החלטות (לשחקנים) |
| game_logic.py | חוקי המשחק (לשופטים) |
| requirements.txt | תלוויות Python |

```

|   └── 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

| קובץ/תיקיה | קורא | כותב | הערות |
|----------------|--------|------------|----------------------|
| config/* | כולם | מנהל מערכת | קריאה בלבד לסטודנטים |
| standings.json | כולם | מנהל ליגה | עדכון אחרי משחק |
| rounds.json | כולם | מנהל ליגה | היסטוריה מחזוריים |
| matches/*.json | כולם | שופט | קובץ למשחק |
| history.json | שחקן | שחקן | היסטוריה אישית |
| logs/* | DevOps | כל סוכן | לוג משלו בלבד |

11.8 התקנה והפעלה

11.8.1 דרישות מקדימות

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

11.8.2 התקנת תלויות

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

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

11.8.3 הפעלת סוכן

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

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

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

11.9 רשימת קבצים מלאה

להלן רשימה מלאה של כל הקבצים בפרויקט:

```
L07/
├─ SHARED/
```

```

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

```

11.10 סיכום

מבנה הפרויקט המוצג בנספח זה מספק:

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

הקבצים המלאים זמינים בתיקייה:

L07/SHARED/

מומלץ לשכפל את המבנה ולהתחליל לבנות את הסוכנים שלהם!

מקורות 12

- 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.