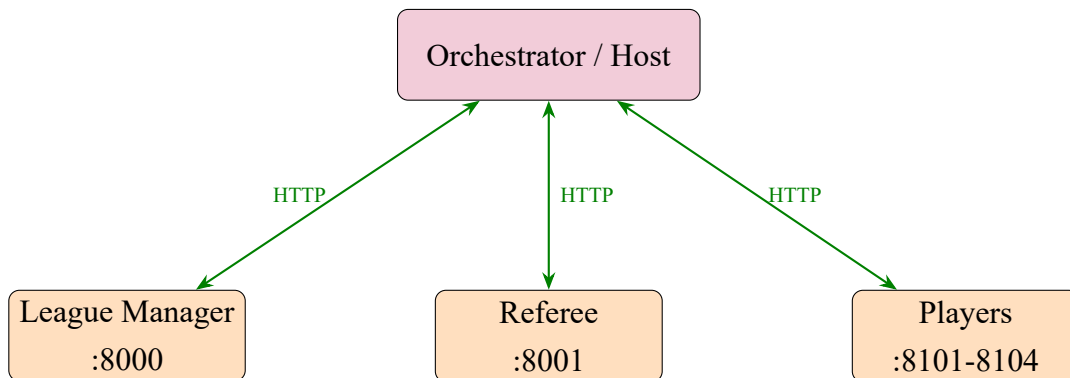


5 מדריך מימוש

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

5.1 ארכיטקטורה כללית

5.1.1 תרשים רכיבים



5.1.2 תפקיד ה-Orchestrator

ה-Orchestrator מתאם בין כל הסוכנים. הוא:

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

5.2 מימוש שרת MCP פשוט

5.2.1 מבנה בסיסי ב-FastAPI

שרת MCP בסיסי

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

app = FastAPI()

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

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

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

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

5.3 מימוש סוכן שחקן

5.3.1 כלים נדרשים

סוכן שחקן חייב לממש את הכלים הבאים:

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

סוכן שחקן פשוט

```

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

app = FastAPI()

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

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

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

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

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

```

5.4 מימוש שופט

5.4.1 כלים נדרשים

השופט חייב לממש:

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

5.4.2 רישום שופט לליגה

שופט נרשם למנהל הליגה

```
import requests

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

קביעת מנצח במשחק זוגי/אי-זוגי

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

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

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

5.5 מימוש מנהל ליגה**5.5.1 כלים נדרשים**

מנהל הליגה חייב לממש:

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

רישום שופט במנהל הליגה

```

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

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

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

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

```

אלגוריתם Round-Robin

```
from itertools import combinations

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

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

    return matches
```

5.6 שליחת בקשות HTTP

5.6.1 קריאה לכלי MCP

שליחת בקשה לשרת MCP

```
import requests

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

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


5.7 ניהול מצב

5.7.1 מצב שחקן

השחקן יכול לשמור מידע פנימי:

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

מחלקת מצב שחקן

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

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

5.8 טיפול בשגיאות

5.8.1 זמן תגובה

בקשה עם timeout

```
import requests

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

5.8.2 תגובה לשגיאות

אם שחקן לא עונה:

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

5.9 דפוסי חוסן (Resilience Patterns)

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

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

לוגיקת ניסיונות חוזרים

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

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

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

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

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

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

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

Circuit Breaker פשוט

```
from datetime import datetime, timedelta

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

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

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

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

5.10 תיעוד מובנה (Structured Logging)

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

טבלה 12: שדות חובה בהודעת לוג

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

5.10.1 מימוש Logger

Logger מובנה

```

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

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

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

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

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

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

```

63

```

    if message_type:

```

```

        log_entry["message_type"] = message_type

```

שימוש ב-Logger

```

logger = StructuredLogger("player:P01")

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

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

```

פלט הלוג:

פלט לוג

```

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

```

5.11 אימות וטוקנים (Authentication)

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

רישום וקבלת טוקן

```

import requests
from dataclasses import dataclass
from typing import Optional

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

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

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

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

```

בקשה עם אימות

```

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

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

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

```


טיפול בשגיאות אימות

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

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

    return True # No auth error
```

5.12 בדיקות מקומיות**5.12.1 הרצה מקומית**

הריצו כל סוכן בטרמינל נפרד:

הרצת הסוכנים

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

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

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

סדר הרצה חשוב:

1. קודם כל מנהל הליגה חייב לרוץ.

2. השופט נרשם למנהל הליגה בעת ההפעלה.
3. השחקנים נרשמים למנהל הליגה.
4. רק אז אפשר להתחיל את הליגה.

5.12.2 בדיקת חיבור

בדיקת שרת

```
import requests

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

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

5.13 טיפים למימוש

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