ODDTi- The Game from school built for Computers

The following is the code for the Game, pls copy paste it to a python interpreter or IDE like Pycharm or Thony, hit Run and start your match…….

Contents

⚡ ODDTI™ v2.3 Predictor Edition — Game Summary

ODDTI™ (read as Oddity) is a digital recreation of one of India's simplest yet most beloved childhood games — Odd/Even Hand Cricket. What began as a casual number game played between friends on classroom benches has been reborn in code, translated into logic, and implemented in modern Python for desktop computers. The idea emerged in 2025 when its creator, Ganesh P. Nair (⚡GPN⚡), was reflecting on his school days during GATE preparation. In a moment of nostalgia fused with engineering thought, he decided to transform the familiar rhythm of odd/even hand cricket into a structured program that merges fun with computational reasoning.

The purpose behind ODDTI was never merely to create a game; it was to revive a memory while teaching logic. The project was guided by three principles — simplicity, authenticity, and accessibility. Simplicity because good engineering is elegant; authenticity because it faithfully recreates a shared childhood experience; and accessibility because anyone can understand how logic governs outcomes. Above all, ODDTI follows a no-jugaad philosophy — every function and feature is written cleanly and deliberately, reflecting a disciplined engineering mindset rather than improvised shortcuts.

At its core, the game replicates the classic hand-cricket rules with computational precision. The player begins with a toss, choosing either odd or even, and then selects a number between 0 and 6. The computer generates its own number, and the sum of both numbers determines who wins the toss. The toss winner then chooses whether to bat or bowl first. In the batting phase, the player inputs a number from 0 to 6 to score runs, while the computer bowls by generating a random number. If both numbers match, the batsman is out; otherwise, the player's chosen number is added to their score. In the bowling phase, the same logic applies in reverse — the computer bats while the player bowls. Once the first innings concludes, the second begins with a target equal to the first side's score plus one. If the chasing side exceeds the target, they win; if they equal it, the match is tied; and if they're out before reaching the target, they lose.

In later updates, the game evolved beyond a single match. The Predictor Edition of ODDTI introduces tournament-style play through a Best-of-Three Series Mode, where players compete over multiple matches to determine an overall champion. Scores are tracked across games, and live updates show totals, targets, and required runs after every ball, creating an engaging and competitive flow. The design was inspired by real-world cricket tournaments and built to bring structure and progression into what was once a purely instinctive game.

What truly sets the Predictor Edition apart is its integration of a lightweight, adaptive logic layer that functions as a basic predictor model. This version allows the computer opponent

to observe patterns in the player's previous inputs and respond with more strategic counterplays instead of relying solely on random generation. Though simple in design, this feature gives the computer a sense of "learning" across matches — subtly mimicking human adaptability. It transforms ODDTI from a fixed-logic game into a dynamic digital opponent that evolves with every round.

The development of ODDTI followed a structured, almost professional workflow. The concept was first visualized through mental flowcharts inspired by C and C++ syntax, before being translated into modular Python code through iterative refinement and AI co-development. Each version of the game was tested and debugged in Thonny and PyCharm, ensuring consistent behavior and stable runtime across different Python environments. Python was chosen deliberately for its clarity and accessibility — a language that allowed the nostalgia of an Indian childhood game to become a real-time demonstration of how logic, probability, and structure interact in programming. Every core function, from choose_odd_or_even() to play_innings() and main(), was written to maintain readability, modularity, and educational value.

The name ODDTI, pronounced Oddity, carries dual symbolism. It represents the game's odd/even foundation while also reflecting the idea of uniqueness — of being the "odd one out." It embodies GPN's philosophy of doing ordinary things uncommonly well: of taking something simple, familiar, and universally Indian, and rebuilding it through clean logic and disciplined creativity. The name mirrors the spirit of the project — playful in heart, precise in execution.

According to the creator, ODDTI is more than software — it is a digital time capsule. It bridges the simplicity of childhood play with the precision of modern computation, preserving culture and memory through code. It represents the belief that engineering isn't about discarding the past but about preserving it in new, meaningful forms. As GPN himself put it:

"ODDTI isn't a project — it's a digital memory capsule.
A bridge between my school bench and my computer screen.
Between India's playgrounds and Python's logic."

From rough sketches and nostalgic inspiration to final, working code, ODDTI™ v2.3 Predictor Edition stands as a product of persistence, curiosity, and structured creativity. It represents how technology can immortalize culture — how a childhood game can evolve into a coded learning tool, a creative expression, and a symbol of disciplined imagination.

In essence, ODDTI is more than just a program; it's a story — a fusion of logic, memory, and identity. It is a reminder that great engineering is not always about invention, but about reimagination — about taking something ordinary and giving it lasting digital form.

———

##Please copy and paste the provided Python code into your preferred Integrated Development Environment (IDE) such as Thonny or PyCharm, or alternatively, save it in a plain text editor and rename the file with a ".py" extension.Predictor model (desktop only)
The "predictor" enhancement in ODDTI™ v2.3 uses standard Python machine-learning libraries and requires Python 3.8 or newer on a desktop environment (Thonny, PyCharm, VSCode). The predictor cannot run on TI-84 Plus CE Python (limited memory and an older Python subset). To use the predictor, run the game from a PC/Mac with Python 3.8+ and install the dependencies listed in requirements.txt. The TI edition (ODDTI2.py) (separate)fully functional on the calculator and does not require the predictor.

#/Code>—

```
# ============================================================
# ⚡ ODDTI™ v2.3 + CPU Predictor — "Odd/Even Hand Cricket"
# Integrated Predictor by ChatGPT for Ganesh P. Nair
# Compatible: Thonny, PyCharm
# ============================================================

import random
import sys

VALID_NUMS = [0, 1, 2, 3, 4, 5, 6]

# ----------------------------
# Predictor: frequency + 1-step Markov
# ----------------------------
# Lightweight, in-memory learning. No external libs.
# Usage:
#   pred = Predictor()
#   pred.predict(last_player_move) -> most likely next player move (0-6)
#   pred.update(prev_move, actual_move) -> update counts after observing actual_move
# ----------------------------
class Predictor:
    def __init__(self):
        # overall frequency counts of player's numbers
        self.freq = {n: 1 for n in VALID_NUMS}  # init with 1 for Laplace smoothing
        # markov: previous -> counts of next
        self.markov = {prev: {n: 1 for n in VALID_NUMS} for prev in VALID_NUMS}
        self.last_player_move = None

    def predict(self, prev=None):
        """
        Predict player's next number given prev (last move).
        Returns an integer 0-6 (most likely).
        Uses markov if prev provided and seen, else overall freq.
        """
        if prev is None:
            prev = self.last_player_move

        if prev is not None and prev in self.markov:
            counts = self.markov[prev]
            # choose highest-count key; break ties randomly
            max_count = max(counts.values())
            candidates = [k for k, v in counts.items() if v == max_count]
            return random.choice(candidates)
        else:
            counts = self.freq
            max_count = max(counts.values())
            candidates = [k for k, v in counts.items() if v == max_count]
            return random.choice(candidates)
```

```
    def update(self, prev, actual):
        """
        Update predictor after we observe actual player's number.
        prev may be None on first seen move.
        """
        if actual not in VALID_NUMS:
            return
        # update frequencies
        self.freq[actual] = self.freq.get(actual, 0) + 1
        # update markov if prev provided
        if prev is not None and prev in self.markov:
            self.markov[prev][actual] = self.markov[prev].get(actual, 0) + 1
        # update last seen
        self.last_player_move = actual

    def reset(self):
        self.__init__()


# Global predictor instance and config
USE_PREDICTOR = True        # toggle predictor on/off
PREDICTOR_EPSILON = 0.12    # probability CPU ignores predictor (randomize)
predictor = Predictor()

# ----------------------------
# Input helpers
# ----------------------------

def input_choice(prompt, options):
    opts = [o.lower() for o in options]
    while True:
        resp = input(prompt).strip().lower()
        if resp in opts:
            return resp
        print("Invalid choice. Options:", ", ".join(options))

def input_int_in_set(prompt, valid_set):
    while True:
        try:
            val = int(input(prompt).strip())
        except ValueError:
            print("Please enter an integer.")
            continue
        if val in valid_set:
            return val
        print("Value must be one of:", sorted(valid_set))

# ----------------------------
# Toss phase
# ----------------------------

def choose_odd_or_even():
    return input_choice("Choose odd or even for toss? (odd/even): ", ("odd", "even"))

def player_choose_number():
    return input_int_in_set("Enter your toss number (0–6): ", set(VALID_NUMS))

def toss(player_parity):
    print("\n--- TOSS ---")
    player_num = player_choose_number()
```

```
    comp_num = random.choice(VALID_NUMS)
    print(f"You: {player_num}, Computer: {comp_num}")
    s = player_num + comp_num
    result_parity = "even" if s % 2 == 0 else "odd"
    print(f"Sum = {s} → {result_parity}")
    if result_parity == player_parity:
        print("You win the toss!")
        return "player"
    print("Computer wins the toss!")
    return "computer"

# ----------------------------
# CPU choice wrappers using predictor
# ----------------------------
def cpu_choose_when_bowling(prev_player_move):
    """
    CPU is bowling (player batting). CPU should try to predict player's chosen bat
    and pick that number to get an OUT. If predictor disabled or epsilon triggers,
    pick random.
    """
    if USE_PREDICTOR and random.random() > PREDICTOR_EPSILON:
        pred = predictor.predict(prev_player_move)
        # choose predicted value (aggressive)
        return pred
    else:
        return random.choice(VALID_NUMS)

def cpu_choose_when_batting(prev_player_move):
    """
    CPU is batting (player bowling). CPU would like to avoid being out:
    predict player's likely bowl, and pick a different number to avoid equality.
    Also bias toward higher scoring numbers for competitiveness.
    """
    if USE_PREDICTOR and random.random() > PREDICTOR_EPSILON:
        pred = predictor.predict(prev_player_move)
        # choose a number != pred, prefer higher numbers but keep some randomness
        candidates = [n for n in VALID_NUMS if n != pred]
        # weight by value: replicate values so larger numbers slightly more likely
        weighted = []
        for n in candidates:
            weight = 1 + n  # simple weight: 1..7
            weighted.extend([n] * weight)
        return random.choice(weighted)
    else:
        return random.choice(VALID_NUMS)

# ----------------------------
# Innings logic (uses predictor)
# ----------------------------

def play_innings(batting, target=None):
    score = 0
    print(f"\n--- {batting.upper()} INNINGS START ---")
    prev_player_move = predictor.last_player_move  # help predictor by providing last seen
    while True:
        if batting == "player":
            p = input_int_in_set("Enter your number (0–6): ", set(VALID_NUMS))
            # CPU selects bowl: try to guess player's number (to get out)
            c = cpu_choose_when_bowling(prev_player_move)
            print(f"Computer bowls: {c}")
```

```
            # update predictor with player's move after we observe it (for next ball)
            predictor.update(prev_player_move, p)
            prev_player_move = p
            if p == c:
                print("You're OUT!")
                break
            # scoring rule per your mod: only player's number counts (you changed earlier)
            score += p
            print(f"Runs this ball: {p} | Total: {score}")
            if target is not None:
                runs_left = target + 1 - score
                if runs_left <= 0:
                    print("Target achieved! 🎯")
                    break
                else:
                    print(f"Runs required: {runs_left}")
        else:
            # Computer is batting
            # CPU selects own bat number (tries to avoid being out using predictor)
            c = cpu_choose_when_batting(prev_player_move)
            # player bowls
            p = input_int_in_set("Enter your bowl number (0–6): ", set(VALID_NUMS))
            print(f"Computer bats: {c}")
            # update predictor with player's bowl (player's action)
            predictor.update(prev_player_move, p)
            prev_player_move = p
            if c == p:
                print("Computer is OUT!")
                break
            # scoring rule: only CPU's number counts when CPU batting
            score += c
            print(f"Computer runs this ball: {c} | Total: {score}")
            if target is not None:
                runs_left = target + 1 - score
                if runs_left <= 0:
                    print("Computer reached the target! 🏏")
                    break
                else:
                    print(f"Computer needs {runs_left} runs more.")
    print(f"--- {batting.upper()} INNINGS END: Score = {score} ---\n")
    return score

# ----------------------------
# Scorecard + match handling
# ----------------------------

def display_scorecard(player_score, computer_score, player_batted_first):
    print("\n" + "="*36)
    if player_batted_first:
        print("Innings order: Player batted first  →  Computer chased")
    else:
        print("Innings order: Computer batted first  →  Player chased")
    print("-"*36)
    print(f"Player score    : {player_score}")
    print(f"Computer score  : {computer_score}")
    print("-"*36)
    if player_score > computer_score:
        print(f"Result: Player wins by {player_score - computer_score} runs.")
    elif computer_score > player_score:
```

```
        print(f"Result: Computer wins by {computer_score - player_score} runs.")
    else:
        print("Result: Match tied!")
    print("="*36 + "\n")

def bat_or_ball_choice_for_player():
    return input_choice("You won toss. Choose to bat or bowl? (bat/bowl): ", ("bat", "bowl"))

def single_match():
    # Toss
    player_parity = choose_odd_or_even()
    toss_winner = toss(player_parity)

    if toss_winner == "player":
        choice = bat_or_ball_choice_for_player()
        player_bats_first = (choice == "bat")
    else:
        comp_choice = random.choice(("bat", "bowl"))
        print(f"Computer chooses to {comp_choice}.")
        player_bats_first = (comp_choice != "bat")

    if player_bats_first:
        print("You bat first.")
        player_first_score = play_innings("player")
        print(f"Computer needs {player_first_score + 1} to win.")
        computer_second_score = play_innings("computer", target=player_first_score)
        player_score = player_first_score
        computer_score = computer_second_score
    else:
        print("Computer bats first.")
        computer_first_score = play_innings("computer")
        print(f"You need {computer_first_score + 1} to win.")
        player_second_score = play_innings("player", target=computer_first_score)
        computer_score = computer_first_score
        player_score = player_second_score

    display_scorecard(player_score, computer_score, player_bats_first)
    return player_score, computer_score, player_bats_first

def best_of_three():
    player_wins = 0
    comp_wins = 0
    match_no = 0

    while match_no < 3 and player_wins < 2 and comp_wins < 2:
        match_no += 1
        print(f"\n=== SERIES: Match {match_no}/3 ===")
        p_score, c_score, p_batted_first = single_match()
        if p_score > c_score:
            player_wins += 1
            print(f"Match {match_no} Winner: Player")
        elif c_score > p_score:
            comp_wins += 1
            print(f"Match {match_no} Winner: Computer")
        else:
            print(f"Match {match_no}: Tie (no points)")
        print(f"Series so far: Player {player_wins} - Computer {comp_wins}")

    print("\n=== SERIES COMPLETE ===")
    if player_wins > comp_wins:
```

```
        print(f"You win the series {player_wins} - {comp_wins}! 🏆")
    elif comp_wins > player_wins:
        print(f"Computer wins the series {comp_wins} - {player_wins}! 🤖")
    else:
        print("Series tied overall.")
    print("========================\n")

# ----------------------------
# CLI helper to manage predictor
# ----------------------------

def predictor_menu():
    global USE_PREDICTOR, PREDICTOR_EPSILON, predictor
    while True:
        print("\n-- Predictor Menu --")
        print(f"(1) Toggle predictor (currently {'ON' if USE_PREDICTOR else 'OFF'})")
        print(f"(2) Epsilon (randomness) = {PREDICTOR_EPSILON:.2f}")
        print("(3) Reset predictor memory")
        print("(4) Show top frequencies")
        print("(5) Back")
        ch = input("Choose: ").strip()
        if ch == "1":
            USE_PREDICTOR = not USE_PREDICTOR
            print("Predictor now", "ON" if USE_PREDICTOR else "OFF")
        elif ch == "2":
            v = input("Enter epsilon (0.0 - 0.9 suggested): ").strip()
            try:
                vv = float(v)
                PREDICTOR_EPSILON = max(0.0, min(0.9, vv))
                print("Epsilon set to", PREDICTOR_EPSILON)
            except:
                print("Invalid number.")
        elif ch == "3":
            predictor.reset()
            print("Predictor memory reset.")
        elif ch == "4":
            print("Overall freq:", predictor.freq)
            # show markov for last seen (if any)
            if predictor.last_player_move is not None:
                print("Markov row for last seen (", predictor.last_player_move, "):",
predictor.markov[predictor.last_player_move])
            else:
                print("No last move yet.")
        else:
            break

# ----------------------------
# Main menu
# ----------------------------

def main():
    print("=== ⚡ ODDTI™ v2.3 (Predictor edition) ===")
    while True:
        print("\n(1) Single Match")
        print("(2) Best of 3 Series")
        print("(3) Predictor Menu")
        print("(4) Quit")
        choice = input("Choose (1/2/3/4): ").strip()
        if choice == "1":
```

```
            single_match()
        elif choice == "2":
            best_of_three()
        elif choice == "3":
            predictor_menu()
        else:
            print("Exiting. ⚡GPN⚡ out.")
            break

# ----------------------------
# Program entry
# ----------------------------

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        print("\nInterrupted. Bye!")
        sys.exit(0)
```

Gameplay and installation manual >—

⚡ ODDTI™ v2.3 — Python Edition (IDE compatibility only)

Designed by: Ganesh P. Nair (⚡GPN⚡)

Engine & Implementation: ChatGPT (Co-developer)

Build: Python Edition v2.3

———

🎯 Game Concept

ODDTI™ (read as Oddity) is the modern digital recreation of India's favorite school-time game — Odd/Even Hand-Cricket.
It brings the simplicity of childhood play into a coded, logic-based Python experience designed for desktop systems.

You choose numbers, play a toss, bat, bowl, chase runs, and win matches — just like on the school bench, only this time in digital form.

———

🧠 System Requirements

Platform	Requirements
Computer (macOS / Windows / Linux)	Python 3.8+ (Thonny, IDLE, PyCharm or VS Code)

———

💻 Installation on Desktop / Laptop

1. Copy ODDTI.py into your working directory.
2. Open your preferred Python IDE or terminal.
3. Run:

3b) (follow only if it doesn't start running and its stuck at import prompt-)

```
>>> import ODDTI
>>> ODDTI.main()
```

4. Enjoy the same gameplay experience on your computer screen.

———

🕹️ Gameplay Overview

🏏 ① Toss Phase

- Choose Odd or Even.
- Pick any number between 0–6.
- The computer chooses its number.
- Sum decides the winner:
- If the total parity matches your call → you win toss.
- Otherwise → computer wins toss.

🧩 If you win, you can choose to Bat or Bowl.

🧠 If computer wins, it randomly decides for you.

⚾ 2️⃣ Innings 1

- If you're Batting:
- Enter a number between 0–6.
- Computer bowls (chooses a random number).
- If both match → you're OUT!
- Otherwise → you score the runs you chose.
- If you're Bowling:
- Computer bats (chooses random number).
- You bowl (enter number 0–6).
- If numbers match → Computer OUT!
- Otherwise → computer scores its chosen number.

🧱 3️⃣ Score & Target

- Score updates automatically after every ball.
- Required runs are displayed dynamically when chasing.
- After first innings, the target = previous score + 1.

🏆 4️⃣ Innings 2 & Result

- Chasing side bats.
- Win Conditions:
- Beat target → 🏆 You win
- Equal target → 🤝 Tie
- Out before reaching target → 💥 Lose

📊 Series Mode (Best of 3)

- After each game, choose "Next Match."
- Scores and wins are stored in memory.
- The first to win 2 out of 3 matches becomes Tournament Champion!
- Displays cumulative stats and match summaries.

🔢 Example Gameplay Log

=== Odd/Even Hand-Cricket Game ===
Choose odd or even for toss? odd
Choose a number (0-6): 3
Computer chooses: 4
Sum = 7 (odd) → You win the toss!
You chose to Bat first.

--- PLAYER INNINGS START ---
Enter your bat number: 4
Computer bowls: 2
You scored: 4 runs (Total: 4)
...
Out! Final Score: 24
Target for computer: 25
--- COMPUTER INNINGS START ---
Computer bats: 2
You bowl: 1
Computer scores: 2 (Total: 2)
...
Computer OUT at 18!
🏆 You win by 6 runs!

———

⚙️ ODDTI™ v2.3 — Predictor Edition Update

(Adaptive CPU + Smart Learning Engine)

ODDTI™ v2.3 marks the next evolution of the project — transforming it from a fixed-logic number game into an adaptive, learning-based experience powered by a built-in lightweight Predictor Engine.
This version introduces AI-like behavior while staying fully compatible with both TI-84 Plus CE Python calculators and desktop Python environments.

———

🤖 What's New

1️⃣ Adaptive CPU Intelligence (Predictor Model)
The computer opponent now "learns" your playing style over time.
It analyzes your previous choices — both while batting and bowling — and stores them in a small internal model that estimates what number you're likely to play next.

This Predictor isn't a pre-trained AI; it's a self-learning system written from scratch using pure Python logic. It uses two techniques:

- Frequency Analysis: tracks how often you play each number (0–6).
- Markov Prediction: observes what number you usually play after a certain number, building a transition map of your habits.

As matches progress, the Predictor updates itself dynamically, giving the CPU personality and unpredictability — sometimes defending cleverly, other times attacking directly to get you out.

———

🧠 How It Works Behind the Scenes

Mode	CPU Role	Predictor Behavior

Player Batting	CPU Bowling	Predictor tries to guess your number and pick the same to get you OUT. (Aggressive mode)
Player Bowling	CPU Batting	Predictor tries to avoid your number, preferring different or higher numbers to stay not out. (Defensive mode)

- The Predictor updates its internal table after every ball, meaning the more you play, the smarter it becomes.
- If you change your pattern suddenly, the CPU will take a few turns to adapt again — just like a real opponent learning your play style.

———

⚙️ Tuning the Predictor

Use the Predictor Menu inside the main menu to modify its settings:

(1) Toggle predictor ON/OFF
(2) Adjust “Epsilon” (randomness factor)
(3) Reset predictor memory
(4) View current learned data
(5) Back

- Predictor ON/OFF → lets you switch between classic and learning gameplay.
- Epsilon (default = 0.12) → decides how “human” the CPU feels.
- Lower value = more predictable (serious mode).
- Higher value = more random (fun / arcade mode).
- Reset Memory → clears all learned data, useful if multiple players are using the same calculator.
- View Data → displays what the CPU has “learned” so far about your moves.

———

💾 Technical Design

The Predictor uses a class-based implementation:

```
class Predictor:
    def __init__(self):
        self.freq = {0:1,1:1,2:1,3:1,4:1,5:1,6:1}
        self.markov = {n:{m:1 for m in range(7)} for n in range(7)}
```

It runs purely in-memory, so it doesn’t require any additional storage or libraries.

———

🏏 Gameplay Impact

- Each ball now feels more alive — the CPU adjusts to your rhythm.
- You can sense increasing difficulty as the series progresses.
- Predictor can be toggled off anytime for pure nostalgic, non-learning gameplay.
- Perfect for demonstrating AI concepts like reinforcement and pattern learning on a simple platform.

🧩 Educational Significance

ODDTI™ v2.3 bridges gaming and machine learning at a level accessible to every student.
It demonstrates how basic algorithms (like frequency analysis and state transitions) can create intelligence without massive datasets or neural networks.

This makes it ideal for classroom demos or personal learning projects — especially for those beginning their journey in Python, AI, or game logic design.

⚡ **Version Summary**

Version	Highlights	Platform
v2.0	Base single match logic	TI-84 / PC
v2.1	Best-of-3 series added	TI-84 / PC
v2.2	Scorecards, runs required, UI improvements	TI-84 / PC
v2.3 Predictor edition (This version)	⚙️ **Predictor AI added, smarter CPU, memory system**	**PC only**

🧠 Developer Note

“The Predictor engine was never about making the CPU unbeatable.
It’s about teaching pattern, logic, and probability —
showing how a simple Python class can simulate intelligence.”

— ⚡ Ganesh P. Nair (GPN)

💡 Pro Tip

If you’re using ODDTI™ for a presentation or portfolio,
highlight the Predictor’s code block and explain:

- how it updates live (like reinforcement learning),
- how its Markov table mimics memory,
- and how it enhances user engagement.

It’s the simplest demonstration of learning behavior in a standalone Python game —
a perfect blend of nostalgia, logic, and AI.

🏁 Summary

ODDTI™ v2.3 — Predictor Edition — turns a nostalgic school game into an intelligent digital companion.
It still plays fair, but now, it learns — one ball at a time.

“From playground rules to predictive algorithms —
ODDTI™ proves that simplicity and intelligence can share the same code.”
———————————————————

🏁 Credits

Role	Contributor
Concept, Design & Game Logic	Ganesh P. Nair (⚡ GPN ⚡)
Core Engine & Implementation	ChatGPT GPT-5
Platform Compatibility	Python 3.x

Version: ODDTI™ v2.3 – Predictor Edition
Tagline :“The classic Indian bench game, reborn in code.”

📘 License & Use

This project is for educational and recreational use, part of GPN’s creative portfolio and engineering learning process.
Free to share or modify — kindly credit original authors.

———

⚡ Summary Command Cheat-Sheet

Platform	Run Command
macOS / Windows / Linux	Import ODDTI → ODDTI.main()