**Hackathon Project Phases Template** for the **Tiktac Game** project.

# **Hackathon Project Phases Template**

## **Project Title:**

## **Intelligent Board Game Opponent and Adviser**

## **Team Name:**

(Play AI)

## **Team Members:**

* N.Pragna
* Poulabi Ghosh
* P.Veronica Sherlyn
* N.Viraja

## **Phase-1:** **Brainstorming & Ideation**

### **Objective:** Generate innovative ideas for an AI-powered board game opponent that adapts to strategies and provides insightful guidance. Explore gameplay mechanics, learning algorithms, and user experience enhancements.

### **Key Points:**

1. **Problem Statement:** Players often struggle to find challenging opponents and receive meaningful strategy guidance in board games. An AI-powered opponent and adviser can enhance gameplay by providing adaptive competition and real-time strategic insights.
2. **Proposed Solution:**

The proposed system is an AI-driven board game opponent that dynamically adjusts its difficulty and playstyle while offering real-time strategy recommendations. It utilizes machine learning and game theory to enhance player experience and skill development.

1. **Target Users:**

The target users include board game enthusiasts, casual players, and competitive gamers seeking challenging AI opponents and strategic guidance. It also benefits learners aiming to improve their skills and developers integrating AI into board games.

1. **Expected Outcome:**
   * A functional **AI-powered Tiktac Game** that provides insights based on real-time data and user queries.

## **Phase-2: Requirement Analysis**

### **Objective:**

Identify the functional and technical requirements for developing an AI-powered board game opponent and adviser. Analyze user needs, AI capabilities, and system performance to ensure an engaging and adaptive gaming experience.

### **Key Points:**

1. **Technical Requirements:**
   * Programming Language: **Python**
   * Backend: **Chat GPT API**
   * Frontend: **Streamlit Web Framework**
   * Database: **Not required initially (API-based queries)**
2. **Functional Requirements:**

* The AI must support and enforce the rules of various board games.
* Validate and reject illegal moves made by players.
  + - Recognize and declare game results (win, loss, or draw).

1. **Constraints & Challenges:**
   * **Processing Power:** AI decision-making, especially for complex games (like Chess or Go), can be computationally expensive.
   * **Memory Usage:** Large game trees and deep learning models require significant memory.
   * **Response Time:** AI must provide real-time move recommendations (<1 sec for simple games, <5 sec for complex ones).

## **Phase-3: Project Design**

### **Objective:**

Develop the architecture and user flow of the application.

+----------------------+

| Start Game |

+----------------------+

|

v

+----------------------+

| Initialize Board |

| & Choose Markers |

+----------------------+

|

v

+----------------------+

| Determine First Player |

+----------------------+

|

+------------+------------+

| |

v v

+----------------+ +----------------+

| Player 1 Turn | | AI Turn |

+----------------+ +----------------+

| |

v v

+----------------+ +----------------+

| Display Board | | AI Calculates |

| Get Input | | Best Move |

| Validate Move | | (Minimax) |

+----------------+ +----------------+

| |

v

### **Key Points:**

### **1. Game Components & Data Values**

| **Component** | **Description** |
| --- | --- |
| **Board** | A list of 9 elements representing the 3x3 grid (e.g., ['X', 'O', ' ', ...]) |
| **Player 1 Marker** | 'X' or 'O' (Chosen at game start) |
| **AI Marker** | Opposite of Player 1’s choice |
| **Turn Order** | Randomly determined (Player 1 or AI starts first) |
| **Win Conditions** | A player wins if they occupy three cells in a row, column, or diagonal |
| **Draw Condition** | If all 9 cells are filled and no one wins |

### **2. AI Decision Making (Minimax Algorithm)**

| **Key AI Concept** | **Value/Role** |
| --- | --- |
| **Scoring System** | AI assigns values to moves: **+1 (AI wins), -1 (Player wins), 0 (Draw)** |
| **Maximizing Player** | AI (tries to get the highest score) |
| **Minimizing Player** | Human Player (tries to get the lowest score for AI) |
| **Best Move Selection** | AI simulates all possible moves and picks the optimal one |

### **3. Game Functions & Their Key Values**

| **Function** | **Key Role** | **Key Values** |
| --- | --- | --- |
| display\_board(board) | Displays current game state | Board values (X, O, ) |
| player\_input() | Asks player to choose X or O | 'X' or 'O' |
| place\_marker(board, marker, position) | Places the player's move | Board index (1-9) |
| win\_check(board, mark) | Checks if a player has won | Winning patterns |
| space\_check(board, position) | Checks if a move is valid | Returns True if space is empty |
| player\_choice(board) | Gets player’s move input | (1-9) with validation |
| ai\_turn(board, ai\_marker, player\_marker) | AI selects best move | Calls minimax() |
| minimax(board, depth, is\_maximizing, ai\_marker, player\_marker) | AI decision algorithm | Returns best move score |

### **4. Game Outcomes & Values**

| **Outcome** | **Condition** |
| --- | --- |
| **Player Wins** | Player has a winning combination |
| **AI Wins** | AI has a winning combination |
| **Draw** | Board is full, no winner |
| **Continue Playing** | No win, spaces still available |

## 

## **Phase-4: Project Planning (Agile Methodologies)**

### **Objective:**

### **1. Project Overview**

**Playful AI** is an AI-driven system designed to play and provide advice for board games such as **Tic-Tac-Toe, Chess, and other strategy games**. The AI will act as an **opponent** (playing against users) and an **advisor** (suggesting best moves).

### **2. Objectives & Goals**

✅ Develop AI-powered board game opponents for various difficulty levels.  
✅ Provide real-time intelligent move recommendations.  
✅ Support both **single-player (against AI)** and **multi-player** modes.  
✅ Use **machine learning** (for advanced games) and **rule-based logic** (for simpler games).  
✅ Ensure an intuitive and engaging user experience.

### **3. Functional Requirements**

#### **Core Features**

| **Feature** | **Description** |
| --- | --- |
| **Game Engine** | Implements game rules, win conditions, and move validation. |
| **AI Opponent** | Uses rule-based logic (Tic-Tac-Toe) and Minimax/ML (Chess) to make decisions. |
| **Move Advisor** | Suggests the best possible move for players. |
| **Difficulty Levels** | Easy, Medium, Hard AI difficulty settings. |
| **Multi-Game Support** | Expandable to multiple board games (e.g., Chess, Connect Four). |
| **User Interface** | Clean UI for move selection and visualizing game states. |

#### **AI Capabilities**

| **AI Feature** | **Description** |
| --- | --- |
| **Minimax Algorithm** | Used for games like Tic-Tac-Toe & Chess (depth-limited pruning for efficiency). |
| **Machine Learning Model** | For advanced strategy games (e.g., reinforcement learning for Chess). |
| **GPT-powered Advisor** | Provides strategic advice in natural language. |

### **4. Technical Requirements**

#### **Frontend Technologies**

* **React.js / HTML5 / JavaScript** – Web interface for playing games.
* **CSS & Tailwind** – For UI styling.
* **WebSockets / Firebase** – For real-time multiplayer interactions.

#### **Backend Technologies**

* **Python (Flask/FastAPI)** – Game logic & AI engine.
* **Node.js (optional)** – Server-side API handling.
* **PostgreSQL / Firebase** – For storing game history & user preferences.
* **OpenAI API** – For AI-generated advice (e.g., Chess strategy suggestions).

#### **AI & Game Algorithms**

* **Rule-based AI** – For simple games (Tic-Tac-Toe, Checkers).
* **Minimax with Alpha-Beta Pruning** – For Chess & similar turn-based games.
* **Reinforcement Learning (Optional)** – For adaptive AI that improves over time.

### **5. Project Timeline (Phases & Milestones)**

| **Phase** | **Tasks** | **Duration** |
| --- | --- | --- |
| **Phase 1: Research & Planning** | Define scope, select board games, research AI approaches. | 2 weeks |
| **Phase 2: Backend Development** | Implement game logic & AI algorithms for Tic-Tac-Toe. | 4 weeks |
| **Phase 3: AI Training** | Train AI models for complex games (e.g., Chess). | 6 weeks |
| **Phase 4: Frontend Development** | Build UI for users to interact with AI opponents. | 4 weeks |
| **Phase 5: Integration & Testing** | Connect frontend & backend, test AI opponent performance. | 3 weeks |
| **Phase 6: Deployment & Launch** | Deploy on web & collect user feedback. | 2 weeks |

### **6. Constraints & Challenges**

| **Challenge** | **Solution** |
| --- | --- |
| **AI Performance Optimization** | Use Minimax with pruning, reinforcement learning for advanced games. |
| **Scalability for Different Games** | Use a modular architecture to support multiple games. |
| **Real-Time AI Response** | Optimize computations & pre-train models to reduce lag. |
| **User Engagement & Experience** | Provide customizable difficulty & interactive AI responses. |

### **7. Risk Management**

| **Risk** | **Mitigation Strategy** |
| --- | --- |
| **AI not providing optimal moves** | Continuous testing, fine-tuning AI algorithms. |
| **High computational costs** | Optimize AI models, use caching techniques. |
| **User frustration with AI difficulty** | Allow users to choose different AI skill levels. |

### **8. Future Enhancements**

🚀 Support for **Chess, Checkers, Go, and other board games**.  
🚀 Implement **voice-enabled AI coaching** for strategy explanations.  
🚀 Develop a **mobile app** for on-the-go board game AI.  
🚀 Introduce **adaptive AI** that learns from user gameplay.

### **9. Tools & Technologies**

* **Programming Languages:** Python, JavaScript (React)
* **AI Frameworks:** TensorFlow, PyTorch
* **APIs & Libraries:** OpenAI API, Flask/FastAPI
* **Database:** PostgreSQL, Firebase
* **Deployment:** Docker, AWS/GCP

### **10. Conclusion**

The **Playful AI - Intelligent Board Game Opponent & Advisor** will create an engaging **AI-powered gaming experience**, providing **strategic gameplay, move suggestions, and multi-game support**. With a structured **AI engine** and **scalable architecture**, it can be extended to support **various board games and learning mechanisms** in the future.

## **Phase-5: Project Development**

### **Objective:**

### **1. Development Approach**

The development of **Playful AI** follows an **Agile methodology** with iterative development, testing, and feedback cycles. The system is designed to support **AI-driven board game opponents** and **real-time advisors** for multiple board games.

### **2. System Architecture**

The architecture consists of **three main layers**:

#### **Frontend (User Interface)**

* Developed using **React.js (or Vue.js) + Tailwind CSS**
* Provides an **interactive game board** for user interaction
* Handles **real-time updates** for AI moves and game state
* Communicates with the backend via **REST APIs or WebSockets**

#### **Backend (Game Logic & AI Engine)**

* Built with **Python (FastAPI / Flask)**
* Handles **game rules, move validation, and AI opponent logic**
* Integrates **AI models** for decision-making
* Stores **game history and analytics**

#### **AI & ML Models**

* **Rule-based AI** (for simple games like Tic-Tac-Toe)
* **Minimax Algorithm with Alpha-Beta Pruning** (for Chess, Checkers)
* **Reinforcement Learning (Deep Q-Learning, Monte Carlo Tree Search)** (for complex games)
* **GPT-4 API for Natural Language Advice**

#### **Database & Cloud Storage**

* **PostgreSQL / Firebase** for storing game states and user data
* **Redis for caching frequently accessed AI computations**
* **Cloud deployment (AWS/GCP)** for scalability

### **3. Development Phases & Milestones**

#### **Phase 1: Research & Planning (2 Weeks)**

✅ Define the **scope, user stories, and requirements**  
✅ Research **AI approaches** for different board games  
✅ Design **system architecture & technology stack**

#### **Phase 2: Backend Development (4 Weeks)**

✅ Implement **Tic-Tac-Toe** logic with **Minimax AI**  
✅ Develop API endpoints for **game moves, validations, and AI responses**  
✅ Create **basic AI opponent models**

#### **Phase 3: AI Model Training (6 Weeks)**

✅ Train **rule-based AI** for classic board games  
✅ Implement **reinforcement learning models** for Chess & complex games  
✅ Optimize **AI performance & response time**

#### **Phase 4: Frontend Development (4 Weeks)**

✅ Develop **interactive game board UI**  
✅ Implement **real-time game updates** with WebSockets  
✅ Integrate **AI-generated move suggestions**

#### **Phase 5: Integration & Testing (3 Weeks)**

✅ Integrate **frontend with backend AI engine**  
✅ Test AI difficulty levels & move accuracy  
✅ Fix **bugs, performance issues, and edge cases**

#### **Phase 6: Deployment & Launch (2 Weeks)**

✅ Deploy to **AWS / Firebase**  
✅ Release **beta version** for user testing  
✅ Gather **feedback & improve AI logic**

### **4. AI Model Implementation**

#### **1. Tic-Tac-Toe AI (Minimax Algorithm)**

* AI **predicts all possible moves**, assigns scores, and chooses the best move
* Uses **Alpha-Beta Pruning** to optimize decision-making

#### **2. Chess AI (Monte Carlo Tree Search / Minimax with Deep Learning)**

* **Tree search algorithm** evaluates potential game states
* **Trained AI model** predicts best moves based on past games

#### **3. AI Advisor (GPT-4 Integration)**

* **Natural Language Processing (NLP)** generates **human-like move suggestions**
* AI **explains strategic decisions** in **plain English**

### **5. Technologies & Tools**

| **Component** | **Technology** |
| --- | --- |
| **Frontend** | React.js, Tailwind CSS, WebSockets |
| **Backend** | FastAPI / Flask, Python |
| **AI Models** | TensorFlow, PyTorch, OpenAI API |
| **Game Logic** | Minimax, Monte Carlo Tree Search, Reinforcement Learning |
| **Database** | PostgreSQL / Firebase |
| **Deployment** | AWS, Docker |

### **6. Challenges & Solutions**

| **Challenge** | **Solution** |
| --- | --- |
| **AI Optimization for Large Games** | Implement **Alpha-Beta Pruning & Deep Learning** |
| **Real-Time AI Response** | Use **caching (Redis), optimized computations** |
| **User Experience & Engagement** | Provide **difficulty levels, interactive hints, and game analytics** |

### **7. Future Enhancements**

🚀 **Multi-Board Game Expansion (Chess, Checkers, Connect Four, etc.)**  
🚀 **Voice-Controlled AI Advisor**  
🚀 **Mobile App for iOS & Android**  
🚀 **Personalized AI Training for Users**

### **8. Conclusion**

The **Playful AI: Intelligent Board Game Opponents & Advisors** project combines **AI-driven gameplay, real-time strategy advice, and multi-game support**. The development roadmap ensures a **scalable, interactive, and engaging gaming experience** powered by AI.

## **Phase-6: Functional & Performance Testing**

### **Objective:**

## **. Functional Testing**

Functional testing ensures that the AI opponents and advisors behave as expected in different board game scenarios.

### **Test Cases for Functional Testing**

| **Test Case ID** | **Test Scenario** | **Expected Outcome** | **Status** |
| --- | --- | --- | --- |
| **FT-01** | AI opponent can make valid moves in Tic-Tac-Toe | AI moves are always legal | ✅ |
| **FT-02** | AI can detect a winning move | AI selects the winning move when available | ✅ |
| **FT-03** | AI prevents player from winning | AI blocks player’s winning move | ✅ |
| **FT-04** | AI suggests an optimal move | AI suggests a move that improves the player’s chances | ✅ |
| **FT-05** | AI makes random moves in easy mode | AI selects moves randomly for easy difficulty | ✅ |
| **FT-06** | AI makes optimal moves in hard mode | AI selects moves using Minimax / RL strategy | ✅ |
| **FT-07** | Detecting game win/loss conditions | Game correctly identifies win/loss scenarios | ✅ |
| **FT-08** | Game UI updates correctly after each move | UI updates accurately with each move made by AI and player | ✅ |
| **FT-09** | AI Advisor (GPT-4) suggests a move | AI advisor provides a valid move suggestion | ✅ |
| **FT-10** | Handling invalid user input | System prevents and warns against invalid inputs | ✅ |

## **2. Performance Testing**

Performance testing ensures that Playful AI runs efficiently under different conditions.

### **2.1 Load Testing**

* **Objective**: Ensure the system handles multiple simultaneous game sessions without lag.
* **Test Strategy**:
  + Simulate **1,000 concurrent users** playing different board games.
  + Measure **response time, CPU/memory usage, and database performance**.
* **Expected Outcome**:
  + AI response time remains **under 500ms** for optimal performance.
  + No significant system slowdowns or crashes.

### **2.2 Stress Testing**

* **Objective**: Evaluate system performance under **extreme conditions**.
* **Test Strategy**:
  + Increase AI move requests beyond **normal limits** (e.g., 10,000 users).
  + Test AI **decision-making speed** under high computational load.
* **Expected Outcome**:
  + The system should remain **stable** and **recover quickly** after stress conditions.

### **2.3 Response Time Testing**

* **Objective**: Ensure AI and advisor responses are **quick and efficient**.
* **Test Strategy**: Measure response times for different operations:

| **Operation** | **Expected Response Time** |
| --- | --- |
| Player Move Processing | ≤100ms |
| AI Move Decision (Minimax) | ≤300ms |
| AI Move Decision (Reinforcement Learning) | ≤500ms |
| AI Advisor Suggestion (GPT-4) | ≤2s |

* **Expected Outcome**:
  + AI opponent **makes moves in real-time**.
  + AI advisor **provides feedback within 2 seconds**.

## **3. Security Testing**

* **Ensure AI cannot be manipulated** (e.g., forced to make illegal moves).
* **Prevent SQL Injection & XSS attacks** in the game database and UI.
* **Secure API endpoints** to prevent unauthorized access.

## **4. Usability Testing**

* **Game UI must be intuitive and responsive** on desktop & mobile.
* **AI difficulty settings should be clear** to users.
* **Game rules and AI suggestions should be easy to understand**.

## **5. Regression Testing**

* **After every new AI improvement, run all functional tests** to ensure no existing functionality breaks.

## **6. Summary**

✅ Functional testing ensures AI opponents **play intelligently** and the UI updates correctly.  
✅ Performance testing ensures **fast response times** and **scalability**.  
✅ Security testing ensures **safe and fair gameplay**.  
✅ Usability testing ensures **a smooth player experience**.

## **Final Submission**

1. **Project Report Based on the templates**
2. **Demo Video (3-5 Minutes)**
3. **GitHub/Code Repository Link**
4. **Presentation**