From Board to Battle: A Development Blueprint for Your Tactical Robot Duel Game

This report provides a comprehensive development plan for creating a playable demo of a tactical board game, inspired by the mechanics of *Pokémon Duel*, featuring custom robot figures. The plan is structured in four parts: a complete deconstruction of the core game mechanics to serve as a design document, a strategy for gathering essential data from the now-defunct original game, a phased roadmap for software development, and a blueprint for engineering a competent AI opponent.

Deconstructing the Blueprint: A Rulebook for Your Robot Duel Game

To successfully create a clone or parody, a foundational understanding of the original's mechanics is essential. The following sections synthesize the rules of *Pokémon Duel* into a definitive rulebook format, providing the necessary blueprint for implementing the game's logic.[1]

The Objective: Conditions for Victory

A duel is won by achieving one of several distinct victory conditions. While direct territorial capture is the primary goal, strategic elimination and temporal pressure provide alternative paths to success.

- Primary Victory Condition: Goal Capture
  The most direct path to victory is to move one of your six figures onto the opponent's designated goal point, located at the far end of the board. The moment a figure occupies this space, the game concludes immediately, and the figure's owner is declared the winner.4
- Secondary Victory Conditions
  Beyond a simple race to the goal, the game incorporates conditions that reward superior board control and punish inaction.
  - **WaitWin (No Legal Moves):** A player achieves a "WaitWin" if their opponent is unable to make any legal moves at the start of their turn. This state is typically

engineered by blocking both of the opponent's entry points and subsequently knocking out all of their figures remaining on the field, leaving them with no figures to move and no way to deploy new ones.[1]
    - **Time Out:** In player-versus-player matches, each combatant is allotted a fixed amount of time (e.g., 5 minutes) for the entire duel. This timer only counts down during a player's own turn. If a player's timer expires, they automatically lose the match. This mechanic is crucial for maintaining pace in competitive play.[1]
    - **Turn Limit:** To prevent indefinite stalemates, a duel has a hard limit of 300 total turns. If this turn count is reached, the game ends in a draw, which is treated as a loss for both players.[3]

The Battlefield: Board Topology and Key Zones

The game is played on a symmetrical field composed of distinct zones, each with specific rules governing figure placement and movement. Understanding this topology is fundamental to strategic play.

- **The Field:** The main playing area consists of 26 interconnected points, or nodes, plus the two goal points at opposite ends. Figures can only move between adjacent points connected by designated lines, known as "routes".[7] The layout is not a simple grid; it features specific choke points and flanking routes that are central to tactical positioning.
- **The Bench:** At the start of the duel, each player's team of six figures resides on their respective Bench, a zone located off the main field of play.[7]
- **Entry Points:** Each player has two Entry Points situated at the corners of their side of the board. A figure on the Bench can only be deployed onto the field by moving to one of these points, costing 1 Movement Point (MP). A figure cannot be deployed onto an Entry Point if it is already occupied by any other figure, friendly or hostile.[7] Gaining control of an opponent's Entry Points is a powerful strategy, as it severely restricts their ability to bring new figures into play.[8]
- **The P.C. (Repair Bay):** When a figure is knocked out in battle, it is moved to the Pokémon Center (P.C.), which can be conceptualized as a repair bay for your robot figures. This zone has a maximum capacity of two figures. If a third figure is knocked out, a queuing mechanic is triggered: the first figure that was sent to the P.C. is immediately moved back to its owner's Bench. However, this returning figure is afflicted with a "Wait" status for one turn, during which it cannot be moved or deployed. The newly knocked-out figure then occupies the now-vacant slot in the P.C..[3] This cycle acts as a crucial tempo and resource management system, penalizing players who lose figures in rapid succession.

The Flow of Battle: Turn Structure and Phases

Player turns are governed by a strict, sequential order of phases. Actions taken in a later phase preclude returning to an earlier one, demanding careful planning and foresight.[3]

1. **Plate Phase:** At the very beginning of the turn, before any figure is moved, the player may use one or more "Plates" (which can be re-themed as "Schematics" or "Subroutines" for your game). These provide powerful, single-use effects for the duration of the duel.[3]
2. **Ability Phase:** After the Plate phase but before movement, the player may activate any figure abilities that require manual triggering. The specific text of an ability determines whether its use ends the turn.[3]
3. **Movement Phase:** The player must select one of their figures on the board and move it along the routes. A figure can move a number of steps up to its assigned Movement Point (MP) value, which ranges from 1 to 3. A player can choose to move fewer steps than their figure's maximum MP.[2]
4. **Battle Phase:** If a figure's movement ends on a point adjacent to an opponent's figure, the active player has the option to initiate a battle. The turn concludes immediately after the battle's resolution. If the player moves adjacent to an opponent but opts not to initiate a battle, their turn also ends.[7]

The Art of Combat: The Spin Wheel System

Combat resolution in *Pokémon Duel* is a probabilistic system centered on a spin wheel, or "Data Disk," unique to each figure. The outcome is not determined by player choice during battle but by a random spin and a rigid hierarchy of move types.[1] The wheel is divided into 96 segments, with the size of each move representing its probability of being selected.

- Move Colors and Priority Hierarchy
  The core of the combat system is a color-coded priority hierarchy that functions like an expanded game of rock-paper-scissors.
  - **Red (Miss):** Represents a failed action. A Red segment automatically loses to any opposing move, with the sole exception of another Red segment, which results in a draw.[2]
  - **White (Attack):** A standard damage-dealing move, defined by a numerical power value. When two White moves (or a White and a Gold move) are compared, the move with the higher damage value wins, knocking out the opposing figure. White moves lose to Purple and Blue moves.[3]
  - **Gold (Priority Attack):** A superior class of attack. It functions identically to a White move when facing another White or Gold move (highest damage wins). However, it possesses priority over Purple moves, meaning a Gold move will always beat a Purple move, regardless of stars or effects.[2] It still loses to Blue moves.
  - **Purple (Special):** These moves represent a wide array of special effects, such as

    inflicting status conditions, displacing figures on the board, or applying buffs/debuffs. They do not have damage values but are instead ranked by a star rating, typically from one (★) to five (★★★★★). A Purple move beats any White move and any Purple move with a lower star rating. It loses to all Gold and Blue moves.[2]
  - **Blue (Defensive):** The highest priority category, representing defensive actions like dodging or protecting. A Blue move beats every other move color (White, Gold, and Purple). An encounter between two Blue moves results in a draw.[3]
- Draw Conditions
  A battle results in a draw, with no effect on either figure, under the following circumstances:
  - Both figures spin a Red (Miss) segment.
  - Both figures spin a Blue (Defensive) segment.
  - Both figures spin White or Gold moves with the exact same damage value.
  - Both figures spin Purple moves with the exact same star rating.[3]

Attacker	**vs. Red (Miss)**	**vs. White**	**vs. Purple**	**vs. Gold**	**vs. Blue**
Red (Miss)	Draw	Opponent Wins	Opponent Wins	Opponent Wins	Opponent Wins
White	Attacker Wins	Higher Damage Wins	Purple Wins	Higher Damage Wins	Blue Wins
Purple	Attacker Wins	Attacker Wins	Higher Stars Win	Gold Wins	Blue Wins
Gold	Attacker Wins	Higher Damage Wins	Attacker Wins	Higher Damage Wins	Blue Wins
Blue	Attacker Wins	Attacker Wins	Attacker Wins	Attacker Wins	Draw

Advanced Tactics and Board States

Beyond basic movement and battling, several key mechanics create deeper strategic layers.

- Surrounding
  A figure is instantly knocked out, without initiating a battle, if it becomes "surrounded." This occurs when an opponent's figures occupy all adjacent, connected points around it, leaving it with no legal moves. This is a powerful tactic for removing high-threat defensive figures without risking a battle.2
- Status Conditions
  Typically inflicted by Purple moves, status conditions are debilitating effects that persist on a figure until it is knocked out or healed. They are a critical tool for weakening powerful opponents.2
  - **Poison:** Reduces the damage of the figure's White and Gold attacks by 20.[7]

- **Badly Poisoned (Noxious):** A more severe version that reduces damage by 40.[7]
- **Paralysis:** At the start of each battle, one of the figure's move segments is randomly turned into a Red (Miss) segment for that spin.[2]
- **Burn:** Reduces the damage of White and Gold attacks by 10 and turns the figure's smallest attack segment into a Miss.[7]
- **Confusion:** When the figure spins its wheel, the result is shifted one segment clockwise from where it landed.[7]
- **Sleep/Frozen:** The afflicted figure is unable to move or initiate an attack. This condition can be removed if an adjacent friendly figure moves next to it, which ends that player's turn.[2]

The game's design is a complex interplay of these systems. The value of a figure is not determined by a single stat, like damage, but by the synergy between its Movement Points, its ability, and the composition of its spin wheel. For example, a figure with 3 MP but a wheel dominated by a large Blue "Dodge" segment is not a primary attacker; it is a "runner" or "scout," designed to rush key positions and survive encounters. Conversely, a 1 MP figure with a massive, high-damage White attack is a "bruiser" or "goalie," a slow-moving area-denial tool. The P.C. cycle introduces a tempo mechanic, punishing reckless attacks and rewarding players who can force favorable trades. When designing the robot figures for the parody, it is crucial to think in terms of these strategic roles, which emerge from the interaction of all mechanics, rather than designing each element in isolation.

The Arsenal: Crafting the Master Research Prompt

A significant challenge in recreating *Pokémon Duel* is the scarcity of centralized data. The game was retired in 2019, and official figure databases are no longer available.[5] Fan-made resources like DuelDB.nl, once a valuable repository, are now defunct.[11] The remaining information is scattered across wikis, forum posts, and community revival projects. Therefore, an AI-powered research approach is the most efficient strategy for data collection.

The Master Research Prompt: A Multi-Part Directive

To ensure the highest quality and most structured output from a large language model, a multi-part, role-playing prompt is required. The following prompt is engineered to direct the AI to act as a specialist, structure its findings logically, and format the data for easy integration into a game engine.

ROLE AND GOAL:
You are a senior game data analyst and archivist. Your mission is to conduct a deep, exhaustive study of the retired mobile game "Pokémon Duel" and compile its mechanics and figure data into a structured, machine-readable format. You must scour all available public

online sources, including Bulbapedia, fan wikis, Reddit archives (r/pokemonduel), and discussions related to revival projects like "Kaeru Duel" to reconstruct this information.
TASK 1: EXPLAIN THE GAME MECHANICS (RULEBOOK FORMAT)
First, synthesize all available information to explain exactly how the game works, as if you were writing an official board game rulebook. Structure your explanation with the following headers:

1. **Objective of the Game:** Detail all known win and loss conditions (Goal Capture, WaitWin, Time Out, Turn Limit).
2. **The Game Board:** Describe the layout, including the number of spaces, entry points, the Bench, and the Pokémon Center (P.C.) and its mechanics.
3. **Turn Structure:** Explain the precise phases of a player's turn (Plate Phase, Movement Phase, Battle Phase).
4. **Combat Resolution:** This is the most critical part. Create a definitive guide to the spin wheel combat system.
   - Detail the five move colors: White, Gold, Purple, Blue, and Red.
   - Explain the strict priority hierarchy (e.g., Blue beats all, Gold beats Purple, etc.).
   - Explain how damage values (for White/Gold) and star ratings (for Purple) resolve ties and determine winners.
   - Create a "Move Interaction Matrix" table that clearly shows the outcome of every possible color-vs-color matchup.
5. **Special Mechanics:** Explain the "Surround" mechanic and provide a comprehensive list of all Status Conditions (Poison, Paralysis, etc.) and their specific effects on a figure.

TASK 2: COMPILE A COMPREHENSIVE FIGURE DATABASE
Second, your primary task is to find and list every Pokémon figure available in the game at the end of its service. There were approximately 596 unique figures, including alternate forms and Mega Evolutions. For each figure, you must provide its complete data in a structured list.
OUTPUT FORMAT FOR EACH FIGURE (Strictly adhere to this):
For each figure, present its data as a JSON object. This is essential for parsing. Here is the template:

JSON

```
{
  "id":,
  "name": "[Figure's Name]",
  "rarity": "",
  "mp": [Movement Points, e.g., 1, 2, or 3],
  "ability": {
    "name": "[Ability Name, or 'None']",
    "description": "[Full text of the ability's effect.]"
  },
  "wheel": [
  {
    "moveName": "[Name of the move]",
    "moveType": "",
    "damage":,
    "stars":,
    "size":,
    "effect": "[Full text description of any additional effects.]"
  },
  {
    "moveName": "[Name of the next move]",
    "moveType": "[...]",
    "damage": null,
    "stars": null,
    "size": [...],
    "effect": "[...]"
  }
 ]
}
```

INSTRUCTIONS FOR DATABASE COMPILATION:

- Search for "Pokémon Duel figure list," "Pokémon Duel database," "Bulbapedia Pokémon Duel," and data from projects like "Kaeru Duel" and the Tabletop Simulator remake.
- The total size of all wheel segments for a single figure must sum to 96. Ensure your data reflects this.
- Provide the data for as many of the 596+ figures as you can find. Prioritize completeness and accuracy. If data for a specific figure is incomplete, note the missing fields.

Structuring Your Figure Data in-Engine

The JSON output from the master prompt is not just for reference; it is a direct blueprint for the data structures within the game engine. By creating a corresponding data class or resource, this information can be imported and utilized seamlessly.
For a solo developer aiming for a quick demo, the choice of game engine significantly impacts development speed. Engines like Unity and Unreal are industry powerhouses but come with steeper learning curves, especially for those less experienced with C# or C++.[12] GameMaker Studio 2 is highly regarded for 2D but involves its own proprietary language and licensing models.[12]
The Godot Engine stands out as an optimal choice for this project. It is completely free and open-source, features a dedicated and intuitive 2D workflow (unlike Unity, where 2D can feel like a subset of its 3D systems), and utilizes GDScript, a scripting language specifically

designed for ease of use and rapid iteration, with a syntax very similar to Python.[14] This combination lowers the barrier to entry and accelerates the prototyping process, making it an ideal environment for a solo developer.
In Godot, this data can be represented by creating a custom Resource. This is a powerful feature that allows the creation of custom data assets directly within the editor, which can then be loaded by game objects.
Example GDScript Resource for Robot Data:

GDScript

```
# RobotData.gd
# This script defines the data structure for a single robot figure.
extends Resource

# --- Basic Info ---
@export var id: int = 0
@export var robot_name: String = "New Robot"
@export var rarity: String = "Common" # Consider using an Enum: enum {Common,
Uncommon, Rare, EX}
@export var mp: int = 1

# --- Ability ---
@export var ability_name: String = "None"
@export_multiline var ability_description: String = ""

# --- Spin Wheel Data ---
# An array of Dictionaries, where each Dictionary is one wheel segment.
@export var wheel_segments: Array
```

With this script, new "RobotData" resources can be created in Godot's filesystem dock. Each resource file becomes a self-contained data asset for one robot, with fields in the Inspector panel that match the JSON structure, allowing for easy data entry and modification.

Building the Prototype: A Phased Development Roadmap

This section outlines a practical, four-phase plan to develop a functional demo. The approach prioritizes building foundational systems first and layering complexity, ensuring a tangible and testable product at each stage.

Phase 1: Foundation - The Digital Board and Data

The first phase involves setting up the project, creating the static elements of the game, and populating it with initial data.

- **Engine Setup:** Initialize a new 2D project in the Godot Engine. The engine's node-based scene structure is well-suited for organizing the game's components.
- **Board Representation (The Graph):** The game board should be implemented as a **graph data structure**, not a simple 2D array. A graph accurately represents the board's topology of nodes (points) and edges (routes).
    1. Create a Tile scene. This scene will represent a single point on the board. Its script (Tile.gd) should contain variables to store its grid coordinates (e.g., var grid_pos = Vector2i(0, 0)) and an array to hold references to its neighbors (var neighbors =).
    2. Create a GameBoard scene. In its script (GameBoard.gd), programmatically instantiate a Tile for each of the 28 points on the board.
    3. Crucially, after instantiating all tiles, iterate through them and populate their neighbors arrays based on the game board's specific layout. For example, the tile at position (0,0) might have neighbors at (0,1) and (1,0). This explicit linking of nodes is what forms the graph and is essential for all future pathfinding and AI logic.[17]
- **Figure Data and Representation:**
    1. Implement the RobotData.gd resource script as detailed in Section 2.
    2. Using the data gathered by the AI prompt, manually create 6-8 RobotData resource files for a balanced set of initial robots. A good starting mix would include a 3 MP runner, a 1 MP goalie, and several 2 MP all-rounders.[19]
    3. Create a Figure scene to visually represent a robot on the board. This scene's script (Figure.gd) will hold a reference to its RobotData resource (@export var data: RobotData). This links the visual piece to its underlying stats.

Phase 2: Core Logic - Turns and Movement

This phase brings the board to life by implementing the turn-based structure and allowing player interaction.

- **Game State Machine:** A state machine is the standard and most robust way to manage the flow of a turn-based game. Create a main game script (e.g., GameManager.gd) that holds the current state. Define states as an enum: enum GameState { PLAYER_TURN, AI_TURN, PLAYER_WINS, AI_WINS }. The GameManager will be responsible for transitioning between these states.[20]
- **Turn Management:**
    1. When the game starts, set the initial state to PLAYER_TURN.
    2. Create functions like start_player_turn() and start_ai_turn().

3. In start_player_turn(), enable player input (e.g., make figures clickable).
4. In start_ai_turn(), disable player input and call the AI's decision-making function.

- **Movement Logic:**
  1. When the player clicks on one of their figures during PLAYER_TURN, the Figure.gd script should emit a signal.
  2. The GameBoard.gd script will listen for this signal. Upon receiving it, it will perform a **Breadth-First Search (BFS)** algorithm starting from the selected figure's tile. BFS is ideal for this scenario as it efficiently finds all nodes within a certain distance on an unweighted graph.
  3. The BFS will return a list of all reachable tiles within the figure's MP value.
  4. Visually highlight these reachable tiles on the board.
  5. If the player then clicks on a valid highlighted tile, move the Figure scene to that tile's position, update the game state to record the figure's new location, and end the player's turn by transitioning the state machine to AI_TURN.

Phase 3: The Heart of the Game - The Combat Loop

This phase implements the central and most unique mechanic of the game: the spin wheel combat.

- **Triggering Combat:** Modify the movement logic. If a player moves a figure to a tile that is adjacent to an opponent-controlled tile, do not immediately end the turn. Instead, display a UI element, such as a "Battle" button, giving the player the choice to initiate combat.
- **The Spin Simulation:**
  1. When combat is initiated, the GameManager will retrieve the wheel_segments array from the RobotData of both the attacking and defending figures.
  2. Write a function get_spin_result(wheel_data). This function will:
     - Generate a random integer between 1 and 96.
     - Iterate through the wheel_data array, cumulatively summing the size of each segment.
     - When the cumulative sum is greater than or equal to the random number, return the current segment's data. This accurately simulates the weighted probabilities of the wheel.
- **Combat Resolution:**
  1. Create a central function, resolve_combat(attacker_move, defender_move).
  2. This function will be a direct implementation of the **Move Color Interaction Matrix** from Section 1. It will use a series of if/elif/else statements to compare the moveType properties of the two moves based on the strict priority hierarchy.
  3. If the priority is the same (e.g., White vs. Gold), it will then compare damage or stars to determine the winner.
  4. The function should return the outcome (e.g., ATTACKER_WINS, DEFENDER_WINS, DRAW).

- **Applying Outcomes:** After resolve_combat returns a result, update the game state accordingly. If a figure lost, move its scene to the P.C. area and update the P.C. queue. Apply any status effects described in the winning move's effect text. Finally, after all outcomes are applied, end the turn by transitioning the game state.

Phase 4: Closing the Loop - Win/Loss Conditions

The final phase for the demo is to implement the logic that concludes the game, creating a complete and replayable loop.

1. **Goal Capture Check:** After every successful movement action (both player and AI), check the figure's new tile. If that tile is the opponent's goal tile, immediately transition the game state to the appropriate win state (PLAYER_WINS or AI_WINS).
2. **WaitWin Check:** At the beginning of each turn (start_player_turn() and start_ai_turn()), perform a check to see if the current player has any possible legal moves. This involves iterating through all of their figures on the board and bench and determining if at least one can move. If no legal moves are found, the other player wins.
3. **Endgame UI:** When the game state transitions to PLAYER_WINS or AI_WINS, display a simple UI screen announcing the winner and providing a "Play Again" button. This button should reset the board and game state to their initial conditions, allowing for a new match to begin.

Engineering an Adversary: A Blueprint for Your Computer Opponent

For a quick demo, creating a deeply complex, multi-turn-predicting AI is unnecessary and time-consuming. A much more practical approach is to build a **heuristic-based AI**. This type of AI evaluates the "goodness" of the board after every possible immediate move it can make and simply chooses the move that results in the best-evaluated board state. This method is computationally cheap and can produce surprisingly effective and strategic behavior.[22]

AI Design Philosophy: The Heuristic Approach

The AI's "brain" will be a single, powerful function: evaluate_board_state(). This function will analyze the entire board and return a single numerical score. By convention, a positive score is advantageous for the AI, and a negative score is advantageous for the player. The AI's goal is always to make a move that maximizes this score.[24]
The intelligence of the AI emerges not from looking far into the future, but from the careful weighting of different strategic factors in the present. A simple weighted sum of key game

state variables is sufficient to create a competent opponent. For example, the value of occupying an enemy entry point must be weighed against the value of knocking out an enemy figure. By assigning numerical weights to these factors, the AI can make informed, quantitative decisions.

The AI's Decision Engine: Scoring the Board

The evaluate_board_state() function calculates its score by summing up weighted values for various strategic elements on the board. The following table provides a recommended starting point for these weights. These values are the most critical part of the AI's design and should be tuned through playtesting to achieve the desired difficulty and behavior.

Heuristic (Board State Condition)	**Score Weight**	**Rationale**
AI Figure on Player's Goal	+10000	An instant win; this should always be the highest priority.
Player Figure on AI's Goal	-10000	An instant loss; must be avoided at all costs.
AI Figure on Player's Entry Point	+200	High strategic value; restricts opponent's deployments.
Player Figure on AI's Entry Point	-500	High defensive priority; must be removed quickly.
Figure Advantage (per figure)	+100	Each extra figure the AI has on the board is an advantage.
Figure Proximity to Enemy Goal (per step closer)	+10	Encourages aggressive, forward movement.
Figure Proximity to Own Goal (per step closer)	-5	Discourages retreating unless strategically necessary.
Threatening a Knockout	+50	Values moves that initiate a favorable battle.
Being Threatened with Knockout	-75	Values moves that retreat from an unfavorable battle.

The AI's Turn Cycle (The "Think" Loop)

The AI executes its turn by following a clear, deterministic process on each of its turns:

1. **Generate All Possible Moves:** The AI first identifies every legal action it can take. It iterates through each of its figures on the board. For each figure, it determines all

reachable empty tiles (Move actions) and all adjacent enemy figures (Attack actions). This generates a comprehensive list of PotentialActions.
2. **Simulate and Evaluate Each Move:** The AI then loops through this list of PotentialActions. For each action, it performs the following simulation:
   - It creates a temporary, hypothetical copy of the current game state.
   - It applies the action to this temporary state. For a "Move" action, it simply updates the figure's position. For an "Attack" action, a simple approach for the demo is to assume an average outcome or, even more simply, to evaluate the board as if the attack was a guaranteed success. This avoids the complexity of simulating probabilistic combat for every possible move.
   - It calls the evaluate_board_state() function on this new hypothetical board state, using the weights from the table above.
   - It stores the resulting score with the corresponding PotentialAction.
3. **Select the Best Move:** After simulating and scoring every possible action, the AI reviews its list of scored actions. It identifies the PotentialAction that yielded the highest score. This is its chosen move.
4. **Execute the Move:** The AI performs the chosen action on the *actual* game board. Its turn is now complete, and the game state transitions back to the player's turn.

This heuristic-driven process provides a robust and scalable foundation for the computer opponent. By adjusting the weights in the evaluation table, the AI's personality can be fine-tuned—increasing the weight for "Figure Proximity to Enemy Goal" will create a more aggressive AI, while increasing the weight for defending its own entry points will create a more defensive one. This simple, data-driven approach is the most efficient path to developing a challenging and engaging opponent for the game demo.

Works cited

1. Pokémon Duel - Wikipedia, accessed October 10, 2025, https://en.wikipedia.org/wiki/Pok%C3%A9mon_Duel
2. Pokémon Duel Guide - PokéCommunity Daily, accessed October 10, 2025, https://daily.pokecommunity.com/2017/01/24/pokemon-duel-guide/
3. r/pokemonduel Guide: Mastering the Game - Reddit, accessed October 9, 2025, https://www.reddit.com/r/pokemonduel/wiki/guide/
4. Pokemon Duel PART 1 Gameplay Walkthrough - iOS / Android - YouTube, accessed October 9, 2025, https://www.youtube.com/watch?v=00AKtbrH9Co
5. Pokémon Duel | Video Games & Apps - Pokemon.com, accessed October 9, 2025, https://www.pokemon.com/us/pokemon-video-games/pokemon-duel
6. Things Pokemon Duel Doesn't Tell You - IGN, accessed October 10, 2025, https://www.ign.com/wikis/pokemon-duel/Things_Pokemon_Duel_Doesn't_Tell_You
7. Prime Yourself for Pokémon Duels | Pokemon.com, accessed October 10, 2025, https://www.pokemon.com/us/strategy/prime-yourself-for-pokemon-duels
8. Tips and Tricks - Pokemon Duel Guide - IGN, accessed October 9, 2025, https://www.ign.com/wikis/pokemon-duel/Tips_and_Tricks

9. gurish165/Pokemon_Duel - GitHub, accessed October 9, 2025, https://github.com/gurish165/Pokemon_Duel
10. 'Pokémon Duel' Beginner's Guide: How to Play, Best Pokémon, Tips For Wins, Evolution & Fusion - Player One, accessed October 10, 2025, https://www.player.one/pokemon-duel-beginners-guide-how-play-best-pokemon-tips-wins-evolution-fusion-580892
11. accessed December 31, 1969, http://dueldb.nl/
12. Best Game Engines for Beginner Game Developers in 2024 - Game Design Skills, accessed October 10, 2025, https://gamedesignskills.com/game-development/video-game-engines/
13. Unity vs. Godot, pros and cons of each? Which is better for an absolute beginner? - Reddit, accessed October 10, 2025, https://www.reddit.com/r/gamedev/comments/1fxd33a/unity_vs_godot_pros_and_cons_of_each_which_is/
14. Best 2D Game Engines - Career Karma, accessed October 10, 2025, https://careerkarma.com/blog/2d-game-engines/
15. Would you choose Unity or Godot for an RTS game? - Quora, accessed October 10, 2025, https://www.quora.com/Would-you-choose-Unity-or-Godot-for-an-RTS-game
16. Godot vs Unity: Which One Suits You Best in 2025 | Kevuru Games, accessed October 10, 2025, https://kevurugames.com/blog/godot-vs-unity-which-one-suits-you-best/
17. How do I represent a game board using a graph? - Stack Overflow, accessed October 10, 2025, https://stackoverflow.com/questions/29353820/how-do-i-represent-a-game-board-using-a-graph
18. Graph-Based Pathfinding Using C# in Unity - Faramira, accessed October 10, 2025, https://faramira.com/graph-based-pathfinding-using-c-in-unity/
19. Top Tips to Dominate Pokémon Duel | Pokemon.com, accessed October 10, 2025, https://www.pokemon.com/us/strategy/top-tips-to-dominate-pokemon-duel
20. Creating turn-based games | Apple Developer Documentation, accessed October 10, 2025, https://developer.apple.com/documentation/GameKit/creating-turn-based-games
21. How to implement turn based combat? - Questions - Defold game engine forum, accessed October 10, 2025, https://forum.defold.com/t/how-to-implement-turn-based-combat/79946
22. How to do AI for turn-based game : r/gamedev - Reddit, accessed October 10, 2025, https://www.reddit.com/r/gamedev/comments/t0c5d1/how_to_do_ai_for_turnbased_game/
23. AI opponent for board (video) game. How to get started? : r/MLQuestions - Reddit, accessed October 10, 2025, https://www.reddit.com/r/MLQuestions/comments/1hp17sg/ai_opponent_for_board_video_game_how_to_get/

24. Introduction to Evaluation Function of Minimax Algorithm in Game Theory - GeeksforGeeks, accessed October 10, 2025, https://www.geeksforgeeks.org/dsa/introduction-to-evaluation-function-of-minimax-algorithm-in-game-theory/