ISBN: 978-628-96613-3-0. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2026.1.1.2799

Interactive War-tactics based Web Game API implementing AI rules

Sammy El Rowmeim[1]; Steven Ramirez Limon[1]; Sanaa Faquir[1]
[1]Vaughn College of Aeronautics and Technology, New York, USA, *sammyelrowmeim.elrowmeim@vaughn.edu, steven.ramirezlimon@vaughn.edu, sanaa.faquir@vaughn.edu*

Abstract*– Traditional board games and classic games like chess have long emphasized strategic movement and mental acuity, yet they often lack the realistic battlefield dynamics and fluid environmental shifts found in modern simulations. Players seeking a deeper tactical experience often turn into complex strategy games, which can be overwhelming. While traditional game design methods focus on creating immersive experiences through static rules, the integration of web Application Programming Interfaces (API) and artificial intelligence (AI) has ushered in a new era where developers can design games that adapt dynamically to a player's behavior and preferences. This project focuses on creating a novel, browser-based strategy game designed to bridge the gap between classic board games and complex tactical simulations. Dominion: Fields of Honor, a web API, chess-like strategy game, combines the mechanics of the "Dominion" deck-building card game with a "Fields of Honor" concept. It integrates war tactics, unit variety, and terrain-based combat while remaining simple to learn and play online. A key innovation is the integration of a simplified, rule-based AI module designed to adapt to gameplay challenges and generate dynamic in-game events, offering a more personalized experience than traditional static games. The final product is a fully functional web-based game featuring units with unique stats and randomized terrain, along with a scalable architectural blueprint that demonstrates the potential of AI in enhancing simple interactive applications. By utilizing web development technologies and AI rule-based strategies, the game can be deployed across a single, unified framework. This allows players to run the game seamlessly on various devices—from mobile phones to desktop computers—without the necessity for platform-specific versions or redundant development cycles.

***Keywords*— browser-based game; turn-based strategy; artificial intelligence; terrain combat; web development**

I. Introduction

Over time, advances in technology have been the driving force behind new types of video games. Traditional game design has long been the primary method for creating video games. Developers and designers relied on their own creativity and skills to create games like Street Fighter, Super Mario Bros, and The Legend of Zelda. These classic titles have had a lasting impact, proving the importance of conventional design [1].

As graphics have become more advanced, games have offered more realistic and immersive experiences. Similarly, new control systems, such as the Wiimote and Kinect, have made it possible for players to engage with games through physical movement [2].

Many games were designed using different tools. Some games are based on cards such as "Tiny Islands" [3], an independent browser-based tactics game built with HTML, CSS, and Python. Libraries such as "Chess.js" and "Chessboard.js" [4] were developed and maintained on GitHub and are frequently used by developers recreating chess-like tactics in web browsers. Chessboard.js handles board display and user interaction, while Chess.js manages game rules and logic. Several developers also created open-source browser-based prototypes of the desktop game "Into the Breach" [5], a tactical puzzle-like game where players control mechs to defend against alien monsters. These projects are available on GitHub and were built using Python.

Classic board games like chess focus on strategic movement but lack realistic battlefield dynamics. Players seeking a deeper tactical experience often turn to complex strategy games, which can be overwhelming. Most strategy games also fail to leverage terrain, range, or positioning, and there is a limited number of browser-based tactical games with real-time multiplayer functionality.

Dominion: Fields of Honor aims to bridge that gap with a new, browser-based, chess-like strategy game that integrates war tactics, unit variety, and terrain-based combat—while remaining simple to learn and play online. This project focuses on creating a foundational interactive game experience that runs entirely within a web browser, leveraging front-end web technologies (HTML, CSS, Python) and back-end Python for game logic and server operations. A key innovative aspect of this project is the integration of a simplified, rule-based AI module designed to adapt game challenges, generate dynamic events, and personalize gameplay based on player actions and progress. The aim is to make the game feel more responsive and engaging than a purely static experience, offering a glimpse into how AI can enhance even simple interactive applications.

AI has been actively used in games to design new characters, generate background artwork, and write new scenarios—tasks that traditionally required significant time and cost [6]. Many search algorithms such as minimax and alpha beta pruning were used previously in the design of competitive games. The Minimax algorithm was used in [7] where agents objective was to maximize their own utility while assuming the opponent will act to minimize it. The alpha-beta pruning was also introduced for path optimization to prune all branches that cannot possibly influence the final decision as indicated in [8]. in 1997, a highly optimized version of alpha-beta search developed by IBM's Deep Blue to defeat world champion Garry Kasparov [9] made these strategies more famous when applied. Even today, these algorithms still remain central to state-of-the-art engines like Stockfish, where the alpha-beta pruning concept was merged with iterative deepening and heuristics to achieve superhuman performance levels as mentioned in [10] and [11]. While high-branching games such as Go used Monte Carlo Tree Search

(MCTS), Alpha-Beta pruning remains much better in "tactically heavy" games containing shallow traps because its full-width search avoids the sampling errors inherent in MCTS [12].
The game combines chess-like turns with battlefield elements: units have HP, attack damage, and tactical roles; maps use randomized terrain (hills, rivers, forests, walls); and combat involves range, cover, and flanking mechanics.
As part of following the 2026 game development trend where web game APIs considers the interoperability quality metric important to allow players to use games across diverse hardware ecosystems ranging from mobile devices laptops and desktop browsers [13], our game was designed as a modern 2D Cross-Device (2xD) web API. This allows building a single, unified codebase using standard web technologies that can be seamlessly containerized with Docker. The container can then be deployed to any web server or cloud provider so that players can enjoy the game across mobile and desktop browsers without the need for platform-specific versions [14], [15].
Upon successful realization of the web game API, the following outcomes were anticipated: (1) a fully functional 2D Cross-Device web-based game API with graphics and tools; (2) a robust game logic backend; (3) demonstration of simplified AI integration; (4) an interactive user interface; and (5) a scalable architecture blueprint.

II. Methodology

A. Game Objectives

1) Specific Goals: The goal was to build a fully playable tactical game in the browser featuring impactful terrain and unit abilities that affect battle outcomes, with straightforward controls and broad accessibility.

2) Measurable Goals: Success was defined by the ability to complete a full match smoothly within 25 minutes, ensuring balanced gameplay and no major bugs.

3) Achievable Scope: Developed with client-side technologies over 10 weeks, Dominion avoided server-side complexity to focus on refined gameplay and UI using HTML and Python.

4) Relevance: Dominion: Fields of Honor blends software development, game strategy, and user experience design into a compact, accessible tactical duel that encourages repeated play and strategic thinking.

B. Game Design

The game was designed around the following core elements: a 10x10 game board with terrain that affects combat; 6 unique unit types with health, range, and attack values; full online multiplayer functionality; win condition logic and UI; and a play-tested, bug-fixed version ready for deployment.

C. Game Perimeter

1) In-Scope: Online 1v1 turn-based gameplay; terrain generation and effects; unit actions (move, attack, support); UI for board, units, and stats; and victory logic based on Command Post capture or Commander elimination.

2) Out-of-Scope: Multi-device support or mobile version; advanced graphics and animations.

3) Target Audience: Strategy gamers, students, chess fans, and developers.

D. Game Strategies and Mechanisms

A player wins by either: (1) capturing the enemy's Command Post (the tile at the enemy base), or (2) eliminating all enemy Commander units.

1) *Game Setup*

The game is played on a 10x10 grid by 2 players. Each player controls 1 Command Post (static tile), 2 Commander units, and 8 Standard units drawn from 5-unit types. Units are placed in the player's back two rows at the start.

2) Terrain System

When the game starts, the board is randomly populated with terrain types as shown in Table I. Terrain is chosen using a fixed seed to ensure balance (approximately 20% of the map, spread evenly).

TABLE I
Terrain Information

Terrain Type	Effect	Visual
Plain	No effect	Light beige
Forest	+1 defense for infantry, -1 range for archers	Dark green
Hill	+1 attack range for ranged units	Brown with peaks
River	Reduces movement by 1 and -1 defense for all units	Blue stripes
Mountains	Blocks line of sight and movement	Gray brick

3) Unit System

Each player controls a squad of 10 units: 2 Commanders (which must be protected) and 8 units chosen randomly from the types shown in Table II.

TABLE II
Units Information

Unit	Role	HP	Move	Atk Range	Dmg	Special Ability
Infantry	Standard soldier	3	2	1 (melee)	1	+1 DEF in Forest
Archer	Ranged support	2	2	3	1	Fires over obstacles
Cavalry	Fast attack	4	4	1 (melee)	2	Charge: +1 dmg if moved 3+ tiles
Siege Engine	Heavy artillery	5	1	3	2	Destroys Walls
Commander	Leader	6	2	1 (melee)	3	High damage; must be protected

E. Player Interface

Each player sees the full 10x10 board with terrain overlay, their own units with HP bars. A sidebar displays a turn indicator, unit stats, action options (Move, Attack, Wait), and a mini-map or command log.

F. AI Integration

While AI systems can be a valuable tool for exploring new game design ideas, they are not a complete solution. AI machine learning techniques require a specific context to function and make appropriate decisions—a context provided by the game designer who creates the game world and sets the rules within which the AI must operate [2]. For this reason, several rule-based reactive agent strategies were implemented in targeted use cases, as described in Table III. these structured rules are gradually integrated into proactive governance platforms to facilitate the auditability and transparency that purely probabilistic machine learning models often lack [16]. Below is an example of an example of one IF-TEN rule that was used to test if the enemy agent is close to the opponent castle then it should take priority to capture the castle before any other moves.

```
if dist_to_player_castle == 1:
    castle_terrain =
gs.terrain[gs.player_castle.row][gs.player_castle.col]
            occupied = gs.get_unit_at(gs.player_castle.row,
gs.player_castle.col)
    if castle_terrain != "mountains" and not occupied:
            score += 1000
            plan["type"] = "moveToPlayerCastle"
```

More rules were developed to guide the computer agent to make moves based on the condition it finds itself in.

TABLE III
AI USE CASES IN THE GAME

AI Use Case	Description
Single-Player Mode	The AI controls one player and makes strategic decisions each turn (which unit to move or attack) based on the current game state.
AI Decision-Making	Rule-based logic evaluates unit health, position, terrain effects, and enemy proximity to decide on actions intelligently.
Unit Prioritization	The AI assigns priority values to enemy units (e.g., Commanders over infantry) and targets them based on threat level and vulnerability.

G. Functional Specifications

1) Key Features: The game is a single-player experience against a computer opponent, featuring grid-based movement and combat on a 10x10 board. The UI displays unit information, health bars, and action buttons. Win/loss logic is based on capturing the Command Post or eliminating all Commanders.

2) User Requirements: The game uses keyboard and mouse input with click-to-move controls and has minimal hardware requirements to support broad accessibility.

3) User Workflow: The player opens the game, enters a nickname and an age (must be greater than 8). Once ready, the game begins with turn-based alternating moves, ending when a victory condition is met.

4) Data Inputs/Outputs: Player inputs include selecting units and clicking tiles to move or attack. Outputs include real-time game state updates: unit movements, damage calculations, health adjustments, and unit defeats.

5) Other Systems: The game includes a player records page where users can see each other's usernames and other information. Moreover, it also includes a game guide page to help new players understand the game and an FAQ page for additional information and lastly the about page which has brief information about the developers.

III. RESULTS AND DISCUSSIONS

Developing a game is a complex task that requires multiple skills which include art, design, programming, and engineering. To create a successful game, developers must consider a wide range of criteria that can be broadly categorized into gameplay, technical, and maybe commercial aspects.

A. Testing and Evaluation Phase

Testing and evaluating software engineering projects requires a set of approaches that considers a set of tests for functionality and accuracy purposes as well as other metrics that evaluates the quality of the project.

In the testing phase, we implemented and run different types of tests such as load and stress testing to check if the game API can support multiple concurrent connections while maintaining low latency under peak traffic. integration testing was also applied to verify how both the game engine and the backend database communicates while the API is running. Two more tests such as the documentation usability and the security were also developed to control and verify unauthorized accesses and manipulation and ensure that the API is robust, scalable, and developer-friendly and can prevent exploits.

TABLE IV
WEB-BASED API GAME EVALUATION CRITERIA

Criteria	Testing Results
Performance	All game actions function as intended, including unit movement, combat interactions, terrain effects, and turn logic. Validation was performed using unit testing to verify individual game components and performance testing to ensure smooth gameplay and stable execution during full matches. The system's modular design allows individual components to be tested and maintained independently, while efficient logic handling ensures fast response times during user interactions. Additionally, the use of consistent design patterns contributes to clean code organization, improved readability, and long-term maintainability.
Functionality	This measures the game completeness every time without stopping or generating errors
Security	The game implements basic authentication through login and sign-up pages, with player data stored in a database table containing user records. User passwords are securely stored using password hashing, preventing plain-text password exposure. Administrative users have permission to edit or remove player records, while standard players are limited to view-only access. This role-based access control helps protect player information and maintain data integrity.
Reliability	The game runs on a Web API, ensuring stable execution during gameplay without crashes. The use of a requirements.txt file simplifies dependency management and allows for smooth and consistent installation across environments. Additionally, the application can be easily containerized into a Docker image, enabling reliable deployment and portability across different systems.

Maintainability	The code is modular and well organized across design, interface, and logic components, with clear comments throughout. Continuous refactoring and the use of design patterns improve maintainability and extensibility, while the web server and Docker ensure the database remains accessible across different devices and environments.
Portability	The API can run the same way on any browser (Chrome, Firefox, Safari, Edge). Also, there is no need for different versions, the same version can run and display the same pages on mobile devices as well as desktop or laptop computers.

The evaluation phase is software engineering relies on standardized frameworks to ensure objectivity and quality. The eight main characteristics defined in the SQuaRE (Software Product Quality Requirements and Evaluation) are: functionality, suitability, usability, reliability, performance, efficiency, compatibility, security, maintainability, and portability [17][18]. For our web-based API, we focused mainly on the factors that can evaluate the quality and success of a project, as they address the user's experience and the system's effectiveness in a real-world environment. From all of the eight factors listed above, we choose six factors varying from design phase to maintenance phase following the project development life cycle as shown in table IV: performance, Functionality, security, reliability, maintainability and portability.

Table V shows the measurement we took while running the API. Results showed that our API meets the target values for all the metrics listed in Table IV.

TABLE V
API WEB GAME EVALUATION RESULTS

Criteria	Target Value	Actual Value
Performance	<2.5s	2-3s
Functionality	100%	98%
Security	100%	100%
Reliability	>20 hours	>20 hours
Maintainability	<5 hours	4 hours
Portability	100%	100%

B. Game API

The project's development relied on a diverse set of technical tools and programming languages to build the game API and achieve its core functionality. Python programming language, Flask, HTML, CSS and Jinja were used to implement the API and all the routing pages and methods. Various python libraries were used to implement the related forms and apply security measures such as hashing users' passwords. Other libraries were also used to handle sessions and requests. For the Backend, SQLite was used to store data related to users including the scores. SQLAlchemy was also used with python to interact with the database and implement queries as follows: users are allowed to update their own information while the admin can apply all the CRUD operations on the users table. By leveraging the lightweight Flask framework alongside SQLAlchemy for robust ORM-based database management, APIs are more scalable and therefore, easily containerized using Docker or served via high-performance production servers like Tornado and GUnicorn to ensure concurrent, non-blocking delivery across diverse network environments [19], [20]. For project management and team collaboration, tools such as Trello, Git, and Miro were dedicated to managing various project phases including the design phase.

The API has two actors as shown in Fig. 1: a user or player and an admin. The admin has full control on the API and is allowed to view and manage all user's accounts such as modification and suppression.

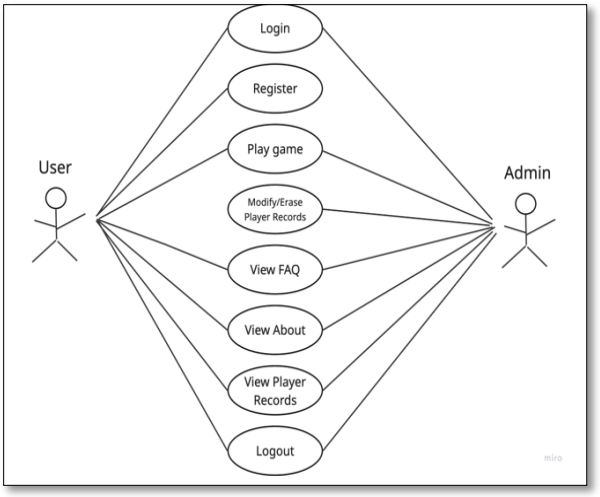

Fig. 1. Use case Diagram showing the actors of the API

Users are only allowed to create a new account (signup) as shown in Fig. 3, login as shown in Fig. 2 or edit their own account as shown in Fig. 5.

The following figures provide a visual overview of the project's main features, demonstrating the final output and user interface.

To ensure a secure environment, our API requires all players to create an account as shown in Fig. 2 and authenticate with their credentials before gaining access as shown in Fig.3. All passwords are scrambled into secure hashing codes before being stored in the backend database for security purposes.

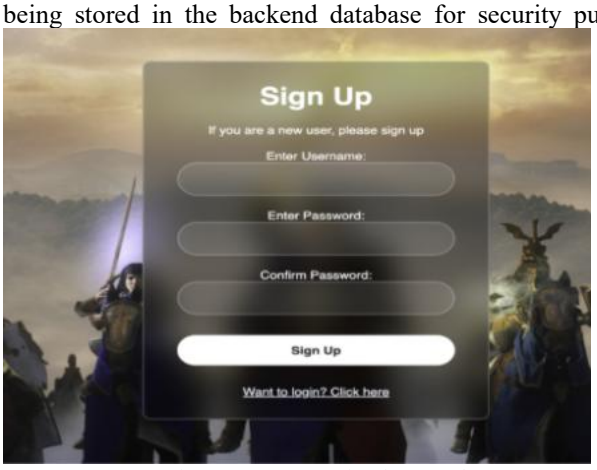

Fig. 2. Account creation for new players

The API also handles errors such as controlling the account creation for one time only and checking the role for loggers as admins or players.

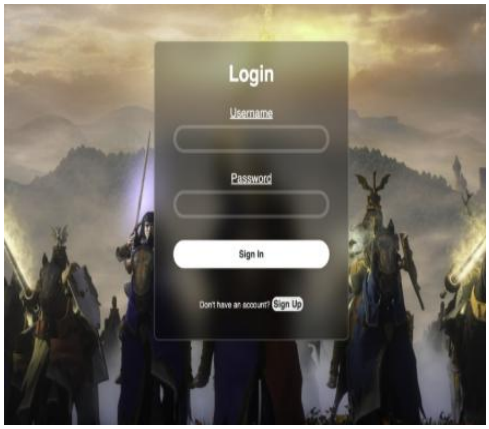

Fig. 3. Players authentication page

The API starts with a starting page that features a centralized menu designed to provide smooth navigation to essential game controls and information as shown in Fig. 4. The menu serves as a user-friendly hub that streamlines game management and support throughout gameplay. By centralizing controls and resources, it reduces confusion and ensures players can navigate easily. Players can find helpful resources such as the Game Guide, about page, player's records, FAQ section or logout from the game. The inclusion of these web pages is a helpful guide to users. The interface is designed to be straightforward and intuitive, ensuring that both new and returning users can navigate the options with ease. Clearly labeled buttons and an organized layout enhance the overall user experience from the moment the game starts.

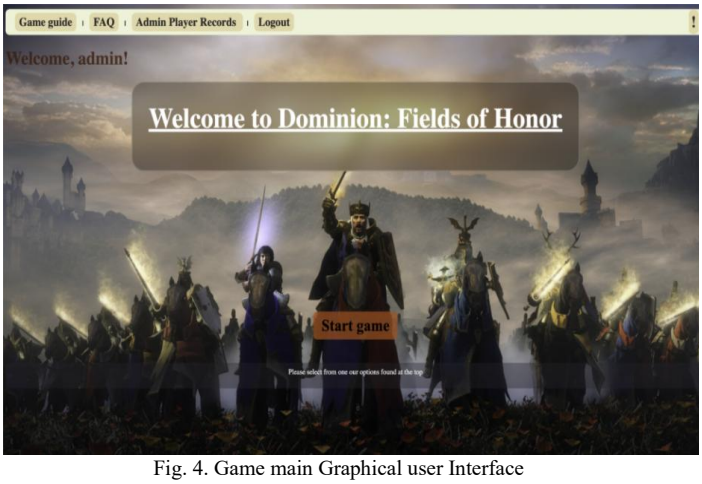

Fig. 4. Game main Graphical user Interface

The backend database stores the users' table information to allow a dynamic communication with the API while the API is running. As mentioned earlier, a user is able to view all other users accounts but can only can edit its own account (changing the user name or the age). For this reason, users are getting the edit button enabled as shown in Fig. 5.

Fig. 5. Players record showing all players information and an edit button enabled

The game starts with a 2D grid as shown in Fig. 6 with the units bar below displaying a variety of unit types that the player can choose from.

Fig. 6. Main Game board with units displayed

Players can then move the units on the grid and position them thoughtfully on the back two rows of the battlefield, avoiding impassable mountain terrain as shown in Fig. 7. Many rules were implemented to make this game succeed such as not being able to place units on mountain terrain. This initial setup emphasizes tactical planning and terrain utilization, setting the stage for an engaging and competitive 1v1 match.

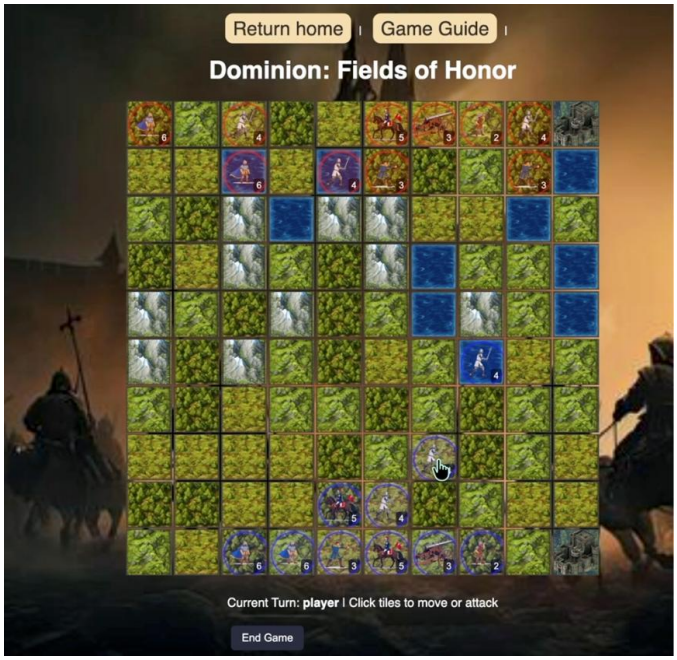

Fig. 7. A battle screen showing the combat phase

After you have placed all 10 of your units on the board within the last two rows, you can proceed by clicking the "Start Battle" button to begin the match as shown in Fig. 7. This initiates the combat phase, where you face off against an intelligent enemy implementing AI game strategies to decide about the best strategically positioned units. During the battle, users will alternate turns, making tactical decisions such as moving units, attacking, or supporting allies, while considering terrain effects and unit abilities. The goal is to outmaneuver and defeat the AI agent playing against them by capturing its Command Post or eliminating all its Commanders.

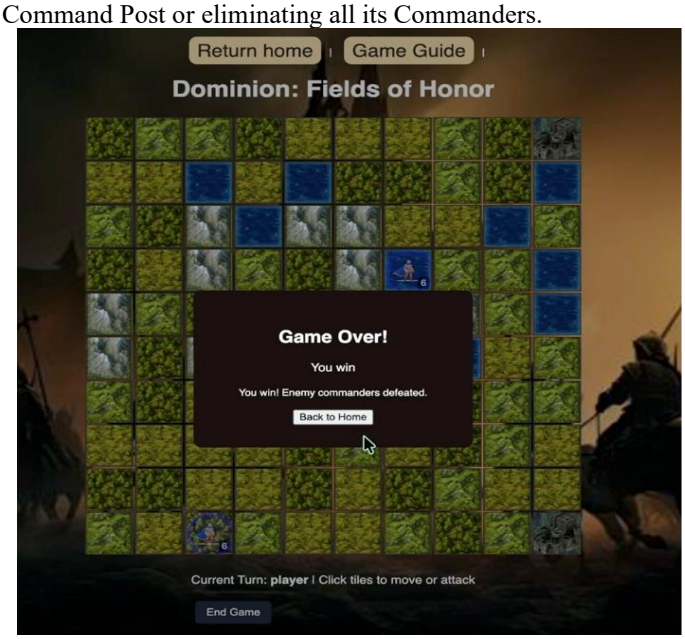

Fig. 8. Screen showing the players victory against the agent

The transition from the setup phase to active battle in Dominion: Fields of Honor marks the start of an intense tactical duel. By combining careful unit placement with strategic combat decisions, players engage in dynamic gameplay that challenges their ability to adapt and outthink the AI opponent.

The game challenges players to leverage terrain effects, unit abilities, and positioning to achieve one of the two conditions to win: capturing the enemy's Command Post as shown in Fig. 8 or eliminating all enemy Commanders.

Many drawing tools were implemented to allow players to navigate between different pages on the API giving them the possibility to return to the home page or leave the game by logging out from their accounts.

IV. CONCLUSION

This paper describes a robust full-stack API web game was developed. The API game blends classic chess mechanics with modern web technologies such as the Python ecosystem along with docker deployment for containerization. This containerized approach allows for universal deployment across web and cloud platforms, ensuring the game is fully accessible on mobile and desktop without overhead of developing separate platform-specific builds. The use of Flask for light microframe and Jinja2 for dynamic server-side rendering helped create a clean separation with the frontend pages and the backend logic. SQLAlchemy was also used to ensure structured and scalable database interactions. Besides technologies, a Rule-based mechanism was also implemented to find the best moves. These rules serve as the fundamental building block for creating interactive and responsive environments for players. By layering these rules, developers can move beyond simple linear progression to create complex, emergent gameplay where the world reacts dynamically to user decisions. The API game also emphasizes security and advanced coding paradigms by implementing authentication and verification while encrypting users and admin passwords using hashing python methods to maintain a security level inside the database. Furthermore, the development phase employs few design patterns and decorators to enhance modularity, readability and maintainability. This work provided a secure, performant and maintainable, web API that demonstrates a deep understanding of the full development lifecycle.

While working on this project, many challenges were faced such as implementing new technologies and system architectures and integrating them effectively to implement the game's functionality without making it overly complex for new players. Through collaborative research, testing based on established criteria, and continuous guidance, the team successfully addressed these challenges and completed the project.

V. FUTURE WORK

Our project has significant room for improvement; the next step that will be implemented is changing the rule-based AI techniques with AI search strategies such as minimax or alpha-

beta pruning for intelligent decision making. Other features can also be implemented such as adding a questionnaire form to allow users to submit suggestions that can be stored in the database for future analysis. We are also planning to integrate sound effects and background music to enhance the game's atmosphere. More new techniques can be the development of an AI chatbot to manage the FAQ section by providing automated responses to player questions. Additionally, future improvements may include a more realistic combat system with an encirclement mechanism, where a unit's combat effectiveness decreases when surrounded by two or more enemy troops, adding a deeper layer of strategy and realism to the gameplay.

ACKNOWLEDGMENT

We would like to thank Vaughn College of Aeronautics and Technology for providing the academic environment and facilities to engage in this challenging and rewarding project.

We would also like to extend our sincere gratitude to our mentor, Dr. Faquir Sanaa, for her invaluable guidance, patience, and expertise throughout this process. Her insights were crucial in helping us navigate the complexities of this project and realize all the necessary deliverables. This project won a challenge in Hackathon on DeveloperWeek 2026 Conference and Hackathon that took place in San Jose on Feb 21, 2026.

REFERENCES

[1] T.-Y. Ennabili, "A Comparison of Traditional Game Design vs. AI-Driven Game Design," 2023. [Online]. Available: https://urn.fi/URN:NBN:fi:amk-2023121437277

[2] M. P. Eladhari, A. Sullivan, G. Smith, and J. McCoy, "AI-based game design: Enabling new playable experiences," UC Santa Cruz Baskin School of Engineering, Santa Cruz, CA, 2011.

[3] D. King, "Tiny Islands," 2019. [Online]. Available: https://dr-d-king.itch.io/tiny-islands

[4] C. Oakman and J. Hlywa, "Chessboard.js," 2013. [Online]. Available: chessboardjs.com

[5] "Into the Breach," Subset Games, 2018. [Online]. Available: https://store.steampowered.com/app/590380/Into_the_Breach/

[6] J. Lee, S.-Y. Eom, and J. Lee, "Empowering game designers with generative AI," *IADIS Int. J. Comput. Sci. Inf. Syst.*, vol. 18, no. 2, 2023.

[7] Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine*, *41*(314), 256–275. https://doi.org/10.1080/14786445008521796

[8] Knuth, D. E., & Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, *6*(4), 293–326. https://doi.org/10.1016/0004-3702(75)90019-3

[9] Campbell, M., Hoane, A. J., & Hsu, F., Deep Blue. 2002. *Artificial Intelligence*, *134*(1-2), 57–83. https://doi.org/10.1016/S0004-3702(01)00129-1 Cited by: 1842

[10] HackerEarth, Understanding Minimax Algorithm with Alpha-Beta Pruning. 2026. . [Online]. Link: https://www.hackerearth.com/blog/minimax-algorithm-alpha-beta-pruning

[11] Informatika, 2025. Implementation of Alpha-Beta Pruning Algorithm for Backgammon Game Engine. Technical Report. https://informatika.stei.itb.ac.id/

[12] Maastricht University. (2024). Monte-Carlo Tree Search and Minimax Hybrids. Link: https://dke.maastrichtuniversity.nl/m.winands/documents/paper%2049.pdf

[13] Juego Studios, 2025. Emerging trends for modern HTML5 game development in 2026. https://www.juegostudio.com/blog/emerging-trends-for-modern-html5-game-development-in-2025

[14] ADEVS. (2026). A Discrete Event System Simulator: Documentation and Deployment for Web-Based Environments. Arizona Center for Integrative Modeling and Simulation. Link: https://www.acims.arizona.edu/adevs/

[15] WeWeb. (2026, February 12). Building Cross-Device Web Applications: Single-Source Deployment Strategies for Mobile and Desktop. WeWeb Engineering Blog. Link: https://www.weweb.io/blog/cross-device-deployment-2026

[16] Jones Walker LLP. (2026). AI at the table: Governing automation in modern gaming. https://www.joneswalker.com/en/insights/blogs/ai-law-blog/ai-at-the-table-governing-automation-in-modern-gaming.html

[17] ISO/IEC. (2011). Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models (ISO/IEC 25010:2011). International Organization for Standardization.

[18] Patil, S. (2025, May 19). *ISO/IEC 25010 Quality Model*. Emergent Mind. https://www.emergentmind.com/topics/iso-iec-25010-quality-model

[19] Real Python. (2026). Scaling Python APIs: Using Tornado as a WSGI container for Flask applications. https://realpython.com/flask-tornado-docker-deployment/

[20] DigitalOcean. (2026). Deploying Python web applications with Docker: A guide to Flask and SQLAlchemy workflows. https://www.digitalocean.com/community/tutorials/python-docker-flask-sqlalchemy