Transformers for Reinforcement Learning in Strategy Games

PIC2 - Master in Computer Science and Engineering
Instituto Superior Técnico, Universidade de Lisboa

João Miguel Martins Carvalho Guimarães Santos — 103617*
joao.carvalho.santos@tecnico.ulisboa.pt

Advisor(s): Pedro A. Santos, João Dias

Abstract Recent advances in Artificial Intelligence have demonstrated that learning-based agents can achieve strong performance in strategic games without relying on handcrafted heuristics, as shown by systems such as AlphaZero and Diplodocus. However, many modern strategy games remain largely unexplored due to increased structural complexity, including large state spaces, imperfect information, and mechanics involving multiple units occupying the same location.

This project investigates the development of a learning-based AI agent for Hispania, a complex historical strategy game featuring stacked units and multi-agent interactions. The primary focus is on designing state and action representations capable of modeling these mechanics in a way that is compatible with modern deep learning architectures, particularly Transformer-based models, and reinforcement learning methods. By evaluating the proposed approach through simulated gameplay, this work aims to assess whether such representations support effective learning and generalization, and whether the resulting architecture can be adapted to other complex strategy games with minimal modification.

Keywords — Transformer, Reinforcement Learning, Strategy Game

*I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa (https://nape.tecnico.ulisboa.pt/en/apoio-ao-estudante/documentos-importantes/regulamentos-da-universidade-de-lisboa/).

Contents

1 Introduction

Strategy games have frequently been used as controlled environments for studying decision-making and strategic reasoning in Artificial Intelligence (AI) [1], [2]. Their well-defined rules and discrete state spaces make them suitable testbeds for exploring planning, learning, and multi-agent interaction. While recent AI systems have exceeded human performance in classical games, many modern strategy games remain largely unexplored due to their scale, multi-agent interactions, and richer mechanics. This interconnected relationship between games and AI promotes advances in both fields, particularly in strategic reasoning and decision-making.

This project aims to investigate learning-based AI for **Hispania**, a historical strategy game featuring multiple factions, evolving board states, and long-term planning challenges. The central problem is to develop an AI that can learn to play Hispania competitively directly from simulated experience, building on foundational reinforcement learning principles [3] and exploring whether modern Transformer-based models can discover strong strategies without relying on handcrafted heuristics or domain-specific rules.

Developing an AI for Hispania is significant for several reasons. First, it advances research in reinforcement learning for imperfect-information, multi-agent environments, which are less explored than classical perfect-information games like Chess or Go [4]. Second, it provides a testbed for techniques that handle complex state representations, such as stacked units, which are common in strategy games but rarely addressed in existing AI systems. Finally, successful approaches could generalize to other historical or strategic games, contributing to broader AI research in decision-making, planning, and strategy under uncertainty.

Hispania is difficult for standard AI approaches because:

- **Imperfect information and stochasticity:** Game events, such as dice rolls, introduce uncertainty, meaning the AI cannot fully predict outcomes in advance. This makes Hispania an imperfect-information game, which contrasts with many classical AI benchmarks like Chess or Go where perfect information is available.

- **Complex state representation:** Board states may involve multiple pieces occupying the same tile, a situation commonly referred to as *stacking*. Existing AI approaches lack an effective method to represent these stacked states, making it difficult to reason about actions and consequences in the game.

Despite the increase in RL and AI for games, Hispania like games have received little attention, primarily due to the complexity of its game mechanics and strategic depth. Developing effective AI requires substantial computational resources for training, particularly for reinforcement learning models that rely on a large number of simulated games. Additionally, identifying an architecture capable of handling stacked units, partial information presents a significant design challenge. Together, these factors have limited prior exploration in this domain.

This project employs a hierarchical model designed to handle the rich and structured game state of Hispania. The model represents information at multiple levels of abstraction, capturing tile-level, piece-level, and global aspects of the board separately before combining them into a unified game-state representation. This structured approach enables the AI to reason about local and global interactions within the game and is expected to support generalization across different board configurations and gameplay scenarios during training and evaluation.

1.1 Work Objectives

The primary objective of this project is to develop an AI capable of learning to reason about the game state and consistently outperforming random or naive strategies, aiming ultimately to approach or exceed the performance of the current Hispania AI. To achieve this, the project aims to:

- Design a hierarchical architecture composed of separate tile-level, piece-level, and game-state encoders, capable of representing the full complexity of Hispania's board state.

- Train the AI using reinforcement learning on small-scale game scenarios and evaluate whether it can extrapolate to larger, more complex situations.
- Study and assess the AI's performance, analyzing outcomes and comparing them to expected benchmarks or reference strategies.

Overall, this project seeks to provide a proof-of-concept that modern deep learning architectures, combined with reinforcement learning, can successfully tackle historically complex strategy games like Hispania, opening avenues for AI research in rich, multi-agent, and historically grounded environments.

2 Background

Early AI systems for gameplay relied on manually defined rules or scripted behaviors, but recent breakthroughs have been driven by machine learning approaches. One major milestone was the introduction of AlphaGo and later AlphaZero, which demonstrated superhuman performance in complex games such as Go, Chess, and Shogi by combining deep reinforcement learning with Monte Carlo Tree Search (MCTS) [2]. These systems showed that an agent can learn optimal policies directly from gameplay, without human-designed heuristics.

More recently, research has shifted toward Transformer-based architectures, which have achieved state-of-the-art results in language modeling and are increasingly being explored in reasoning and reinforcement learning tasks.

Contemporary board games present new challenges compared to classical perfect-information games. They often include larger search spaces, asymmetric player roles, complex interactions, and situations with incomplete or uncertain information. These challenges require more flexible and scalable AI methods, making them a suitable testbed for modern learning-based agents.

2.1 Hispania

As a concrete testbed for studying these challenges in complex strategy games, this project focuses on **Hispania**, a historical board wargame that exhibits many of the structural properties discussed above, for readers interested in further details about the game make sure to check the appendix. The game features asymmetric factions, imperfect information, stochastic events, and mechanics involving multiple units occupying the same board location, making it a representative and demanding environment for learning-based AI methods.

2.1.1 Overview

Hispania is a strategy game for 1 to 4 players that portrays the struggle for dominance over the Iberian Peninsula. The gameplay covers the period from the Carthaginian invasion in 237 BC to the onset of the Christian Reconquista. Players take command of multiple nations, typically between four and five, which are treated as a single allied faction under the player's control. As a result, interactions between nations controlled by the same player are always cooperative, and hostile actions are only permitted against opposing players.

Units Each nation controls a set of **units** with specific attributes, as shown in Figure 1, that determine their capabilities and importance to the board:

Figure 1: Unit Stats

- **Unit Types:** Units include Infantry, Cavalry, Leaders, and other specialized types. Each type has its own set of attributes that determine its performance in combat and on the map.
- **Hit Points (HP):** Indicates how many hits a unit can sustain before being eliminated.
- **Combat Factors:**
  - **Offensive Factor:** Base value used to calculate successful attacks on enemies.
  - **Defensive Factor:** Base value used to resist attacks from enemies.
  - **Minimum Die to be Killed:** Some units, like Roman legions, require a minimum modified die value to be eliminated. Most units do not have this property.
- **Movement Points (MPs):** The number of tiles a unit can move per turn. Movement is affected by terrain and may be enhanced by leaders.
- **Purchase Cost:** The population points required to build or recruit the unit during the Growth phase.
- **Quantity:** Total number of units of this type available in the scenario, including those already on the map at game start.

Board Once units are available, they are placed on the game board, which is divided into **55** distinct **regions**, as shown in Figure 2. The regions vary in terrain, classified as clear, difficult, or poor, which affects movement and stacking limits. They are grouped into major areas, including the North, Center, South, East, and peripheral off-map zones. Regions may be controlled by a nation, remain neutral, or be occupied by vassals. Some regions contain structures such as forts, castles, or cities that impact defense, income, and stacking rules. Each region also generates population points that can be spent during the Growth phase. The map also features straits, which allow movement between non-adjacent regions, rivers, which can influence combat, and seas, which are accessible only to nations with ships.

At the start of the game, the board is configured with specific units already placed in certain regions for each nation. Within a given region, a player can have multiple units, but only up to a defined **stacking limit**. This limit depends on the characteristics of the region as well as the units and structures present. For example, the presence of a leader or a building in the region can modify the stacking limit, allowing more units to occupy that space.

Figure 2: Game Board

Victory Conditions To win the game, players must accumulate Victory Points (**VPs**), which are earned at the designated VP Count Turns. The player's final score is the sum of the VPs from all the nations they control, and the player with the highest total score is declared the winner.

VPs are awarded based on several factors. Nations earn points for controlling key regions or entire areas, for eliminating enemy units or leaders, and for constructing or maintaining structures such as cities and castles, with capitals and defensive buildings providing extra points. There are also other nation-specific ways to earn VPs, but the methods described above are the standard ways of scoring them.

Gameplay Loop Earning Victory Points requires players to manage their nation's units across regions effectively. The turn order of nations is fixed, and the game typically lasts 21 rounds, each consisting of the arrival of new reinforcements, a complete sequence of nation turns and ending in the VP Count Turn. Each nation's turn is divided into several phases:

- **Growth:** The nation receives population points from controlled regions, which are then spent to build new units. Purchases are mandatory if sufficient points are available. Newly purchased units are immediately usable.
- **Movement:** Units and leaders move according to their Movement Points (MPs). Movement costs depend on the type of terrain. Movement ends when entering difficult terrain or crossing a strait, unless a leader is present.
- **Battle:** All battles initiated during the Movement phase are resolved. Battles consist of consecutive simultaneous-action rounds. A unit scores a hit when rolling a value equal to or greater than its combat strength. The defender decides whether to retreat first, followed by the attacker.

- **Major Invasions:** Certain nations may receive a Major Invasion status, allowing them to move and fight twice in a row within the same turn.
- **Overpopulation:** If a nation has more than twice as many units (excluding leaders and structures) as the number of regions it controls, excess units must be removed before continuing.

2.1.2 Hispania AI

The **Hispania** AI controls non-player nations. As no technical documentation is available, its behavior is described based on gameplay observation. The AI generally follows historically plausible actions, which are often effective given the game's rule design.

The AI's decisions appear to rely on simple, local features of the current game state, such as unit counts in a region, regional control, leader presence, and nearby battles. Movement typically follows historically expected routes or reinforces key regions, while combat decisions focus on resolving immediate conflicts. Rule-specific mechanics such as submissions, stacking limits, and overpopulation are handled conservatively.

Overall, the AI behaves in a reactive manner, with no clear evidence of multi-turn planning or adaptation to unusual situations, due to its heuristics. This limitation motivates the use of learning-based methods to see if they can reason over the full game state and plan across multiple turns.

2.2 AlphaZero

In order to understand the capabilities of modern learning-based AI, it is useful to examine key approaches that have demonstrated strong performance in complex games. One such approach is **AlphaZero**, a general-purpose reinforcement learning algorithm that achieved superhuman performance through self-play, most notably in chess, shogi and go [2].

AlphaZero operates by tightly coupling Monte Carlo Tree Search (MCTS) with a learned policy and value function. Although this search procedure differs from classical MCTS [5], it follows the same core principle of performing a lookahead search from the current game state by repeatedly simulating possible future trajectories. Rather than expanding the search uniformly, the learned policy biases the exploration toward actions that are more likely to be strong, while the learned value provides an estimate of the expected game outcome for newly reached states.

The neural network takes a game state $\mathbf{s}$ as input and outputs a policy $\mathbf{p}$ and a value $\mathbf{v}$,

$$(\mathbf{p}, \mathbf{v}) = f_\theta(\mathbf{s}),$$

where $\mathbf{p}$ is a probability distribution over legal actions and $\mathbf{v} \in [-1, 1]$ estimates the expected outcome of the game from the current player's perspective. These outputs are not used to directly select actions, but instead guide the MCTS process.

Each MCTS simulation consists of four phases: selection, expansion, evaluation, and backpropagation. During selection, actions are chosen according to the Predictor Upper Confidence Bound applied to Trees (PUCT) rule,

$$U(s, a) = C(s) \cdot P(s, a) \frac{\sqrt{N(s)}}{1 + N(s, a)},$$

which balances exploration of less-visited actions with exploitation of actions favored by the policy prior $P(s, a)$. Here, $P(s, a)$ denotes the prior probability of selecting action a in state s, as predicted by the neural network policy head. The term $N(s)$ represents the total number of visits to state s, while $N(s, a)$ denotes the number of times action a has been selected from state s. The coefficient $C(s)$ is an exploration constant that controls the relative importance of the policy prior during search.

When a new state is expanded, it is evaluated by the neural network to obtain a policy prior and a value estimate, which are then propagated back through the search tree to update node statistics.

After many simulations, the visit counts of actions at the root node define an improved policy $\boldsymbol{\pi}$, which is used to select the actual move played in the game. This mechanism allows AlphaZero to approximate

multi-step planning by combining learned evaluations with explicit tree-based search rather than relying on fixed-depth heuristics.

Training is performed through self-play. Games are generated using MCTS-guided action selection, and each encountered state is stored together with the corresponding improved policy $\boldsymbol{\pi}$ and the final game outcome z. These samples are used to train the neural network to better predict both the policy and the value, closing the learning loop.

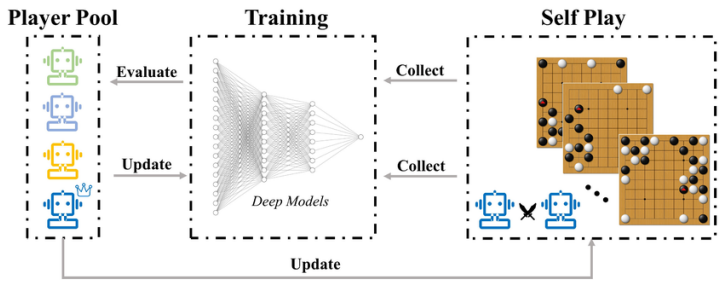


Figure 3: Alphazero pipeline

Through repeated cycles of self-play, search, and training, AlphaZero progressively improves its decision-making, learning both how to evaluate positions and how to guide search effectively without external supervision, as shown in the Figure 3.

2.3 Transformer Encoder

The **Transformer encoder** is a neural network architecture designed to process a sequence of elements, capturing complex dependencies between them through the self-attention mechanism [6]. The encoder processes all elements in parallel and learns their relationships by determining how much attention each element should give to every other element in the sequence. The Transformer encoder architecture is shown in Figure 4.

Input. Before being processed by the encoder, each element is represented as a numerical vector, where each component encodes a specific property of that element. These vectors are then combined into a sequence to form the input of our encoder. Additionally, in order for each element to retain information about its position, a unique vector is added to its representation that is meant to encode its location in the sequence, allowing it to reason about the spatial or sequential relationships between units and regions. These are known as positional encodings.

Self-Attention. The core idea of the encoder is the self-attention mechanism, which determines how much attention each element should pay to every other element in the sequence, allowing the model to weigh the relevance of different elements when building context-aware representations.

In order to do this, for each element three versions of its vector are created. A query vector Q, a vector meant to represent what this element is seeking, a key vector K meant to represent what this element offers to others and a value vector V, what information do i give if paid attention to. These vectors are generated by applying separate linear transformations using weight matrices, which are learned during training.

$$Q = xW_Q, \quad K = xW_K, \quad V = xW_V$$

For each element, the attention mechanism computes a score between its query and the keys of all elements, indicating how relevant each other element is. These scores are scaled by $\sqrt{d_k}$ to maintain numerical stability and passed through a softmax to produce attention weights that sum to 1. The

values of all elements are then multiplied by these weights and summed, producing a new vector for the element that incorporates information from the entire sequence.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

Multi-Head Attention. Instead of performing self-attention just once, the encoder applies it multiple times in parallel using different learned projections. Each parallel instance is called an attention head. Multiple heads allow the model to capture different types of relationships simultaneously: one head may focus on local interactions, another on long-range dependencies, and yet another on structural patterns. The same input is fed to all attention heads, each of which performs self-attention independently. The resulting outputs are then concatenated and passed through a linear projection to produce a single vector per element, integrating information from all heads.

Residual Connection and Normalization. Afterwards, the original input is added back to the output, a process known as a residual connection, and the result is normalized. This helps preserve the original information, stabilizes training, and enables the stacking of multiple layers.

Feedforward Network. Each element is then processed independently by a position-wise feedforward network, which applies the same small neural network to every element. This network performs a non-linear transformation on the element's vector, allowing it to further refine its representation after integrating information from other elements. As with the multi-head attention block, the same residual connection and normalization is applied to the output of this layer.

Output. Everything described above, from multi-head attention to the feedforward network, constitutes a single encoder layer. These layers are stacked N times to progressively refine and enrich the representation of each element. The encoder ultimately outputs a sequence of context-aware embeddings, one for each input element, providing a rich and structured representation of the input. These embeddings are then used by downstream modules, such as decision-making networks, policy heads, or value estimators, in reinforcement learning architectures.

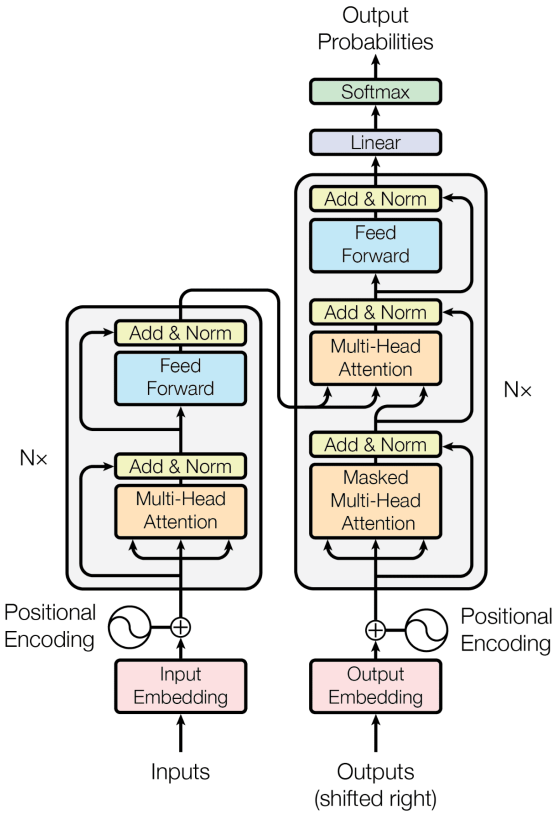


Figure 4: Transformer Architecture taken from [6]

3 Related Work

Research on game-playing AI has evolved significantly over the years, driven by both the increasing complexity of games and advances in algorithms. Early approaches focused on explicit search methods, exploring possible moves and counter-moves to identify optimal strategies. While effective for simpler games, these methods quickly become impractical as the number of possible states grows. To address this, more efficient techniques such as Monte Carlo Tree Search (MCTS) were introduced, which estimate the value of moves through randomized simulations rather than exhaustive evaluation.

In recent years, the integration of deep neural networks with search algorithms has further extended the capabilities of AI agents, enabling them to tackle highly complex strategy games. This section reviews notable systems, highlighting approaches most relevant to AI for games like *Hispania*, with other systems cited in [7], [8], and [9] noted for reference.

3.1 No-Press Diplomacy

No Press Diplomacy is a strategic board game for seven players, each controlling one of the major European powers in the early 20th century. The objective is to capture supply centers and achieve dominance over the continent. The game proceeds in simultaneous turns where players issue orders to move armies and fleets, resolve conflicts, and build new units. Unlike traditional Diplomacy, no-press versions prohibit or communication between players, requiring decisions to be made solely based on the observed game state. Success relies on anticipating opponents' moves, careful planning, and long-term strategy over multiple turns. An overview of the standard game board and territorial layout is shown in Figure 5.

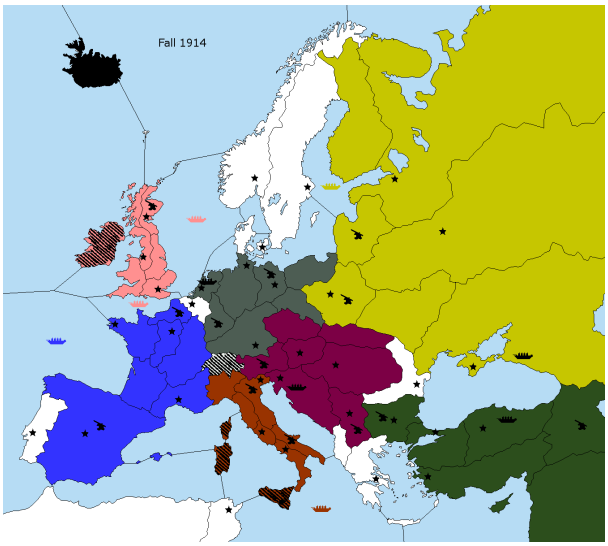


Figure 5: No-press Diplomacy game board

Bakhtin et al. [1] proposed the **Diplodocus** framework, which applies modern reinforcement learning to No-Press Diplomacy while incorporating human alignment mechanisms to improve performance against various play styles.

3.1.1 Model Architecture

Diplodocus uses a Transformer encoder architecture along with a policy and value head, shown in Figure 7. The board state of the no-press Diplomacy game is represented as illustrated in the Figure 6, capturing three types of features: per-location features such as unit type, province ownership, terrain, and recent actions; per-player features including player status, strength, and scoring; and global features such as the current season, year, and other contextual rule settings. These features are first projected into a shared embedding space using learned linear layers, then concatenated to form a single input sequence for the Transformer, with positional encodings added to preserve the order and type of each element.

Feature	Type	Number of Channels
Presence of army/fleet?	Binary	2
Army/fleet owner	One-hot (7 players), or all zero	7
Build turn build/disband	Binary	2
Dislodged army/fleet?	Binary	2
Dislodged unit owner	One-hot (7 players), or all zero	7
Land/coast/water	One-hot	3
Supply center owner	One-hot (7 players), or all zero	8
Home center	One-hot (7 players), or all zero	7

Table 3: Per-location board state input features

Feature	Type	Number of Channels
Number of builds allowed during winter	Float	1

Table 4: Per-player board state input features

Feature	Type	Channels
Season (spring/fall/winter)	One-hot	3
Year (encoded as $(y - 1901)/10$)	Float	1
Game has dialogue?	Binary	1
Scoring system used	One-hot	2

Table 5: Global board state input features

Figure 6: State Representation taken from [1]

These features are first projected into a shared embedding space using learned linear layers, then concatenated to form a single input sequence for the Transformer, with positional encodings added to preserve the order of each element.

3.1.2 Policy and Value Heads

After encoding, the output of the Transformer encoder is separated into two distinct output heads: a policy head responsible for generating unit orders and a value head responsible for estimating long-term game outcomes [10].

Policy Head. Generating a Diplomacy move requires coordinating multiple unit orders that interact through supports, convoys, and movement conflicts. Since these orders are not independent, the policy head must model conditional dependencies across units.

Diplodocus uses an autoregressive LSTM decoder for this purpose. After the Transformer encoder produces contextualized embeddings for every province and global token, these embeddings are pooled into a summarized state representation. This representation initializes the hidden state of the LSTM decoder.

The decoder then generates unit orders sequentially:

1. At each step, the decoder attends to the encoded board state via an attention mechanism, allowing it to incorporate global information.
2. The decoder outputs a probability distribution over all legal actions for the current unit.
3. The selected action is fed back as input into the next LSTM step, enabling conditional structure between unit orders.

In Hispania, actions within each phase are sequential and modify the game state, such that later decisions depend on earlier ones. While an autoregressive LSTM, as used in Diplodocus, is in principle capable of modeling such dependencies through its hidden state, this formulation becomes less suitable for capturing longer-range and more complex interactions between decisions within a phase. For this reason, a Transformer-based decoder is preferred, as it allows direct conditioning on the full sequence of previously selected actions. Nevertheless, Diplodocus's method of encoding the action space for unit movements remains highly applicable and closely aligns with the action representation used in Hispania.

Value Head. The value head provides an estimate of the expected game outcome from the encoded state, which guides both reinforcement learning updates and planning algorithms.

Diplodocus computes the value prediction using an attention based module applied to the set of Transformer outputs. Rather than relying on a fixed pooling operation like mean pooling, the attention mechanism allows the model to focus on strategically relevant parts of the board, such as centers with contested control or heavily concentrated unit clusters.

The resulting aggregated representation is passed through a multilayer perceptron that outputs a scalar estimate of the state's value, a win probability over the most likely player to win at this state. This enables the agent to evaluate long term strategic outcomes, complementing the short term tactical reasoning of the policy head.

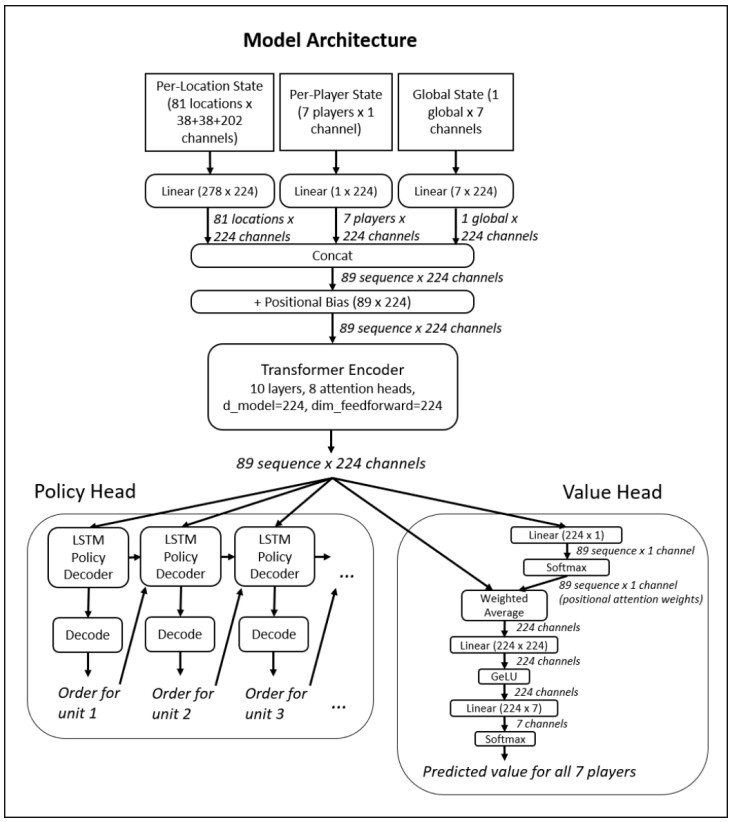


Figure 7: Diplodocus Architecture

3.1.3 Human Aligned Planning Frameworks

In addition to the base model, Diplodocus incorporates several mechanisms to encourage learned policies that resemble human play. The first is **PiKL** (Policy KL-regularization), which penalizes the policy produced by the policy head for deviating from a reference human policy. This approach was originally introduced in the context of regularized search to model strong and human-like gameplay in complex multi-agent settings [11]. A scalar regularization strength controls the trade-off between reward optimization and human alignment. Formally, if π_θ denotes the learned policy and π_H the reference human policy, the KL regularization term is defined as:

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}}(\pi_H \,||\, \pi_\theta),$$

DiL-PiKL extends PiKL by sampling different KL regularization strengths during training, allowing the agent to learn a spectrum of behaviors ranging from highly human like to strongly optimized for winning. This exposes the agent to a variety of strategic styles and improves adaptability.

RL-DiL-PiKL further incorporates this framework into full reinforcement learning self play, combining KL regularization with the RL reward objective. This ensures that the agent not only learns to maximize reward, but also generalizes well against opponents with diverse strategies.

3.2 Settlers of Catan

A related line of work on complex board game state representation is the Deep Reinforcement Learning system developed for *Settlers of Catan* [12]. The proposed Catan AI agent focuses extensively on how to construct an expressive encoding of the full game state before feeding it into a decision making module, as shown by the architecture in Figure 8.

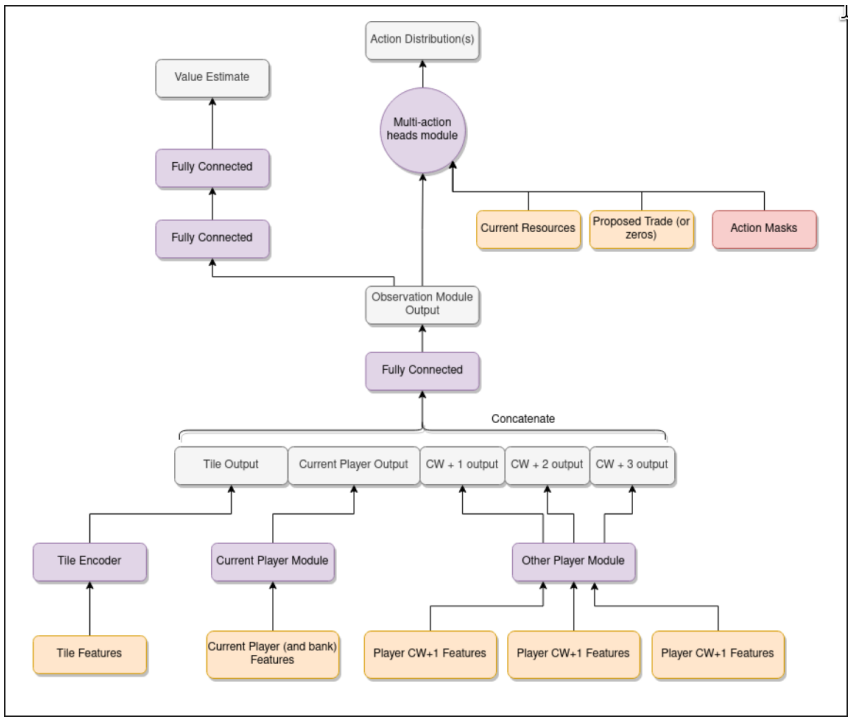


Figure 8: Settlers of Catan AI architecture taken from [12]

3.2.1 State Encoding Architecture

The state space is structured into three primary components: per-tile features capturing the layout of the board, a comprehensive summary of the current player's state, and partial summaries representing the states of other players. These components are then combined into a single, unified representation of the game state that serves as input to the policy learning model.

Tile Encoder. The game's board is represented through the tile encoder, which transforms each board tile into a structured feature vector, as shown in Figure 9. For every tile, several attributes are extracted: the tile value is encoded as a one-hot vector corresponding to the dice number associated with the tile; the resource type is represented with a one-hot encoding specifying the produced resource, such as wood, brick, or ore; a binary robber indicator signals whether the robber currently occupies the tile; and the six adjacent corners are described by features indicating the type of building present (none, settlement, or city) along with the player who owns it.

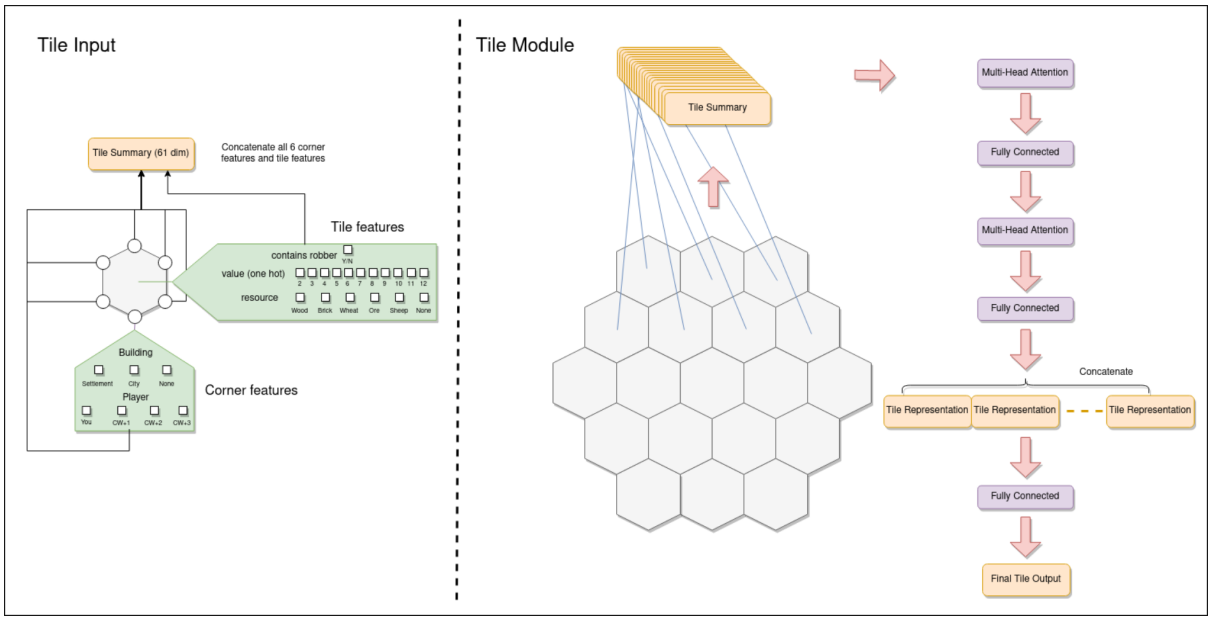


Figure 9: Tile encoder taken from [12]

These attributes are concatenated into a single 61 dimensional vector per tile. The set of 19 tile vectors, total Catan board tiles, is then processed by a stack of alternating multi-head attention layers and fully connected layers. This attention mechanism allows the encoder to integrate global context such as the distribution of high value tiles or resource monopolies while still producing tile specific embeddings.

After the attention stack, the 19 encoded tile vectors are concatenated and passed through a final fully connected layer, yielding the output of the tile module. This constitutes a compact global summary of the spatial resource layout of the board.

Player State Features and Final Concatenation. In parallel, the system encodes player related features. The current player's summary includes information such as resource counts, access to ports, victory points, production potential, and indicators for achievements like largest army or longest road. The other players' states partially observable are encoded as ranges of possible resource counts, visible development cards, and other aggregated statistics. These opponent feature vectors are processed through shared fully connected layers to ensure consistent representation across players.

Finally, the outputs of the tile module, the current player state vector, and the processed opponent vectors are concatenated into a single observation embedding. A final fully connected layer transforms this concatenated vector into the *observation module output*, which serves as the complete encoded game state.

3.2.2 Multi-Head Policy and Action Space

The Catan agent handles the complex action space using multiple heads, each responsible for different components of a player's decision. The first head selects the type of action (e.g., place settlement, play development card), while subsequent heads select specific parameters such as locations, resources, or target players. Conditional dependencies between heads are managed by feeding outputs from earlier heads as inputs to later heads, and invalid actions are masked out, as demonstrated in the interactive simulator [12]. While this approach guarantees correct action sequencing, it introduces a significant computational overhead, as each head must be processed sequentially through the network, making the action selection process slower.

For example, playing a "Year of Plenty" development card is treated as a composite action: one head selects the card, and additional heads choose the two resources. This decomposition allows each

component of the action to be trained independently while still respecting dependencies, improving sample efficiency and generalization.

This approach to multi-head action space representation is highly relevant for games like Hispania, where multiple sequential actions occur within a phase and later decisions depend on earlier ones. By structuring the action space into multiple conditional heads, it is possible to model complex action dependencies without enumerating all possible action sequences.

3.3 Deep RL for Hex-and-Counter Wargames

Hex-and-Counter Wargames are a class of adversarial simulations that feature large hexagonal maps, complex terrain–unit interactions, unit stacking, and simultaneous movement and combat decisions. These characteristics create extremely high-dimensional state and action spaces and require reasoning about spatial relationships, unit interactions, and long sequences of interdependent decisions. Such complexity has motivated the use of deep reinforcement learning, which can learn effective policies directly from structured game states, as demonstrated in recent work on Hex-and-Counter Wargames [13]. An overview of the complete solution is shown in Figure 10, highlighting the extrapolation from training on smaller game boards to testing on larger ones.

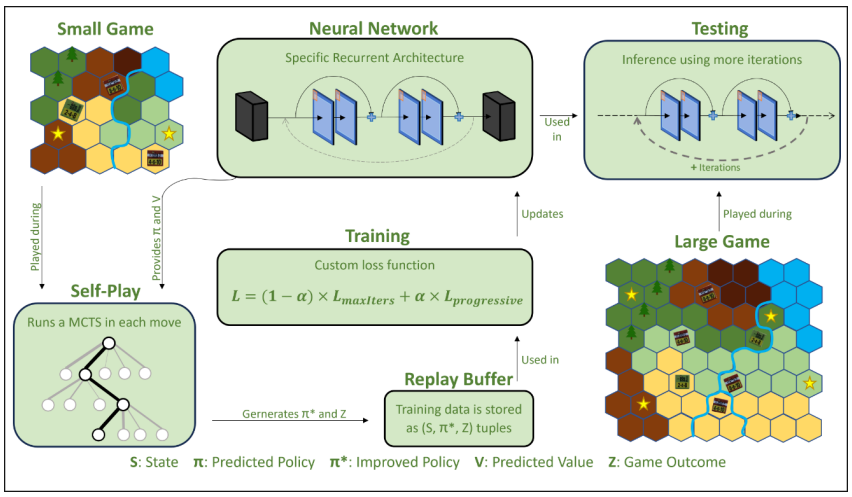


Figure 10: Diagram of the designed solution taken from [13]

3.3.1 State Representation

To handle such complexity, the state representation is designed to generalize across many different wargames without retraining. The complete game state is encoded as a feature stack of shape $H \times W \times C$, matching the spatial resolution of the hexagonal board. Unlike approaches based solely on the current player's perspective, this representation encodes both players symmetrically. All player-dependent attributes are duplicated, ensuring that asymmetric victory conditions, reinforcement schedules, and unit configurations are explicitly captured in the state tensor.

Each tile encodes terrain attributes such as attack modifiers, defense modifiers, and movement costs, as well as any victory-point resources present. Unit stacks are represented with channels for attack strength, defense strength, and movement allowance, up to the maximum stacking limit S. Reinforcements are encoded across R turns to capture future planning possibilities. This results in a final channel count of

$$C = 19S + 12(R + 1),$$

where the channels cover unit stacking ($9S$ per player), reinforcement schedules ($6R$ per player), terrain attributes (3 player-independent channels), and global indicators such as victory conditions.

This state representation encodes both spatial and player-dependent information, allowing the network to reason over long-horizon planning and complex inter-unit interactions. By structuring the state in this way, the model can generalize across different board sizes, unit configurations, and terrain types, making it a flexible template for adapting deep reinforcement learning to large-scale wargames such as Hispania.

4 Solution

The proposed solution tackles the complexity of Hispania using a transformer-based architecture. The design follows the Diplodocus model [1], which demonstrates the benefit of processing the game state through dedicated encoders.

4.1 Solution Overview

Building on this idea, this solution extends the concept by using multiple transformer encoders operating in a staged manner. The first two encoders process different modalities of the game state in parallel: the **Tile Encoder** E_t processes information about the map regions, while the **Piece Encoder** E_p processes information about all pieces currently present on the board. Subsequently, their output is concatenated with the global features of the board and then fed into the **Game-State Encoder** E_g, which produces a unified representation of the entire game configuration.

This design separates tile and piece information, enabling the model to apply distinct attention patterns to each type of feature. The final encoder E_g then operates on per-tile representations that already include all units present in each tile, making the relationship between regions and units explicit in the model.

4.2 Game State Representation and Architecture

The following paragraphs describe how each transformer E_t, E_p, and E_g encodes its corresponding part of the game state.

Tile Encoder E_t. The tile encoder receives a sequence of tokens, one per tile on the board. Each token corresponds to a feature vector that describes a specific province. This feature vector would have the following structure, as shown in Figure 11.

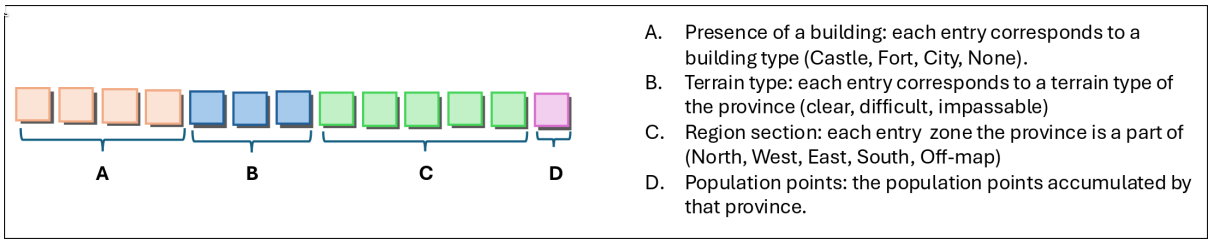


Figure 11: Tile feature vector

Thus, each tile is represented by a feature vector composed of a 4D one-hot (building), 3D one-hot (terrain), 5D one-hot (region), and a float (population points). Therefore, the complete tile-encoding input matrix has shape $N_t \times T_f$, number of tiles ×, number of tile features (55×13), resulting in 715 scalars, as illustrated in Figure 13.

Because the Hispania board is more graph-like rather than a 2D grid, tiles do not possess implicit spatial coordinates. To provide position information, each tile receives a **learnable absolute positional embedding** that identifies that region. However, absolute embeddings alone do not encode adjacency.

To incorporate the map structure, we additionally introduce a **relative positional encoding** implemented as a bias term in the tile-transformer attention mechanism:

$$\text{AttentionBias}(i, j) = f(\text{distance}(i, j)),$$

where distances come from the fixed adjacency graph of the Hispania map. This encourages adjacent regions to be more closely related to each other.

Piece Encoder E_p. The piece encoder operates analogously, but instead of tiles, it encodes every piece currently present in the game state. Each piece is represented by a feature vector as shown in Figure 12:

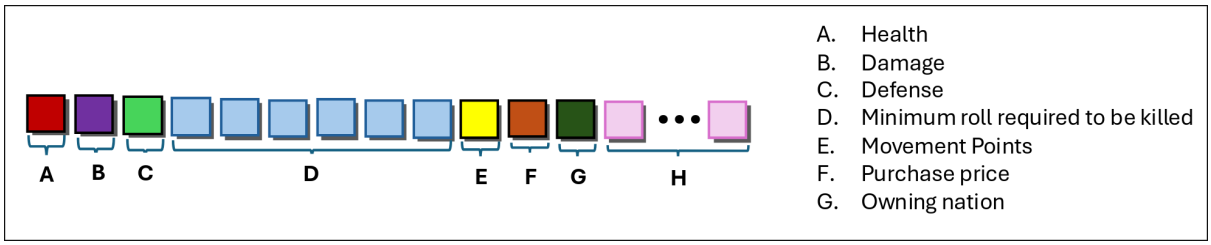


Figure 12: Piece feature vector

Thus, each piece is represented by a feature vector composed of scalar values for health, damage, defense, movement points, and purchase price, a 6-dimensional one-hot vector encoding the minimum die roll required for elimination, a binary flag indicating whether the unit is a leader, and a 20-dimensional one-hot vector encoding the owning nation. Therefore, the complete piece-encoding input matrix has shape $N_p \times P_f$, number of pieces $\times$ number of piece features (N_p, 32), as illustrated in Figure 13.

Unlike tiles, pieces require explicit grounding to the board. To provide this, every piece feature vector receives an additional **tile-reference embedding** obtained from a learnable table

$$E_{\text{tile-ref}} \in \mathbb{R}^{N_{\text{tiles}} \times d_{\text{model}}},$$

which is indexed by the tile on which the piece is currently located. These parameters are trained end-to-end via backpropagation. This mechanism differs from the absolute positional embeddings used for tiles, though both serve to localize tokens within the game map.

Game-State Encoder E_g. The final transformer encoder E_g integrates tile-level and piece-level information. Its input consists of structured per-tile sequences formed as follows:

1. A global feature vector **g_f** is prepared, containing:
   - a one-hot encoding of the current player (4-dimensional).
   - a one-hot encoding of the active nation for the current turn(20-dimensional).
   - a one-hot encoding of the current turn type (5-dimensional).
   - the turn number (float).
   - active population points (float).
   - accumulated victory points (float).
2. For each tile, we concatenate:

$$[\text{global features (g\_f)},\ \text{tile encoding from } E_t(t\_e),\ \text{piece stack for that tile}].$$

3. The **piece stack** for tile i is defined as:

$$\text{Tile}_i^{\text{stack}} = [p_{e1}, p_{e2}, \ldots, p_{ej}, \text{padding}, \ldots],$$

where p_{ej} are the outputs of E_p for all pieces located on tile i. The sequence is padded up to a fixed **maximum stacking limit** $S_{\max}$, meaning it is filled with 0's.

Although the game rules allow leaders to exceed stacking limits and certain exceptional situations can create temporarily higher tile densities, a fixed upper bound is required for transformer processing. We therefore choose $S_{\max} = 6$, which safely exceeds the maximum regular stacking size (4–5 in normal play).

The final input to E_g is therefore a tensor of shape:

$$N_{\text{tiles}} \times (g_f + t_e + S_{\max} p_e),$$

representing the fused state of each tile along with, all pieces located on that tile. The game-state encoder processes this structure to produce the complete latent representation of the board, as shown in Figure 13. For the current configuration, this corresponds to a per-tile input vector of length 237 scalars (6 global features + 13 tile features + 6 pieces × 32 features per piece). With 55 tiles, the full board representation contains 13,035 scalars.

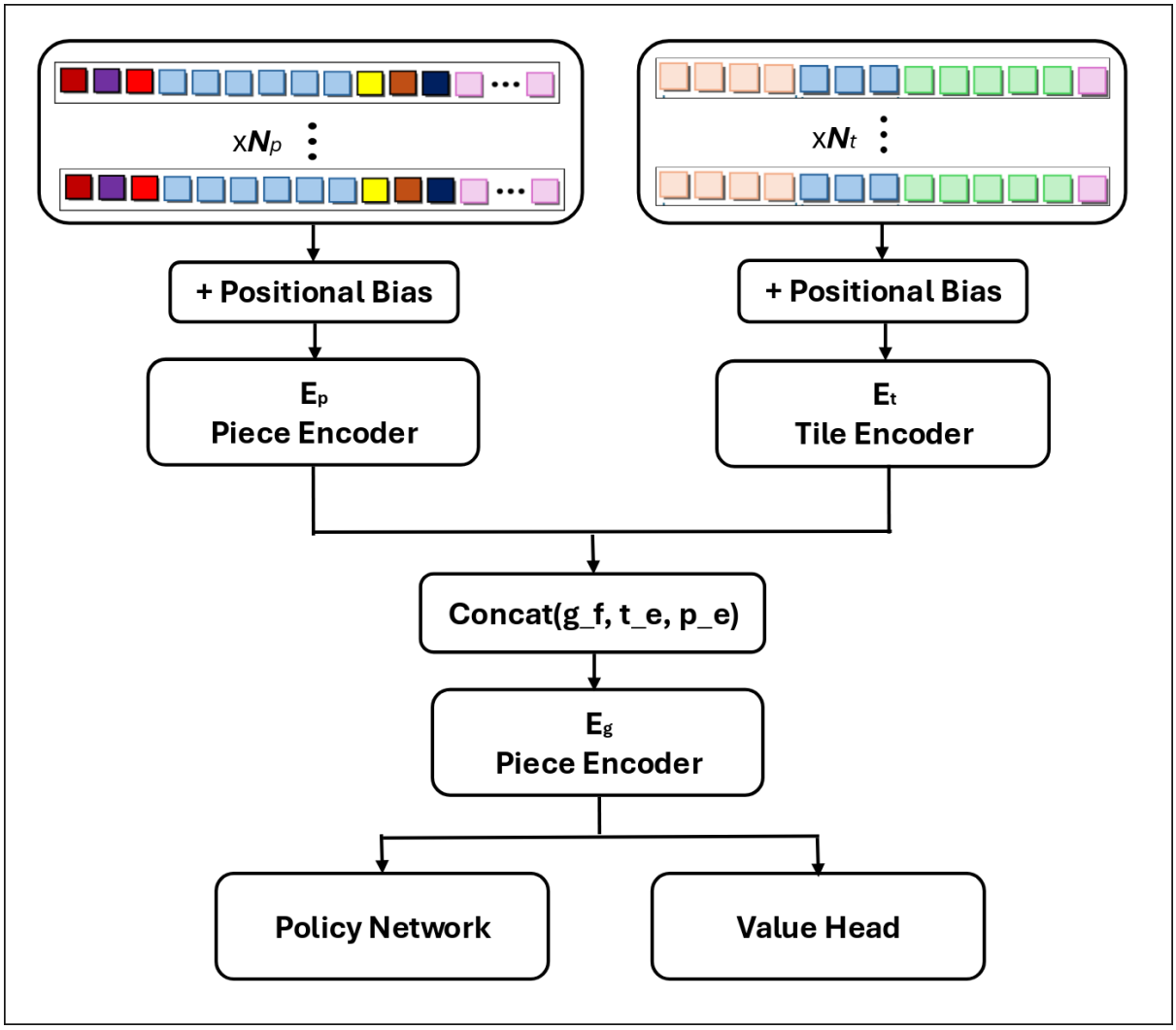


Figure 13: Model Architecture

4.3 Action Space Representation

In Hispania, the agent does not take a single action per turn. Instead, each agent turn corresponds to a single nation's turn, which unfolds across multiple phases. Within each phase, the agent may generate a sequence of one or more decision steps, where each step represents a specific controllable choice for that nation. Random outcomes arising from game mechanics, such as combat resolution, are handled by the environment dynamics and are not part of the agent's action space.

Each decision step is represented as an action token composed of an action type and a set of associated parameters. These action types directly correspond to the phases of a nation's turn described in Section 2.1.1: Growth, Movement, Battle, and Overpopulation. The Major Invasion phase is treated analogously to the Battle phase, as it involves repeated movement and combat decisions within the same nation turn.

Each decision is encoded as a structured vector, referred to as an action token, which represents only the agent's choice at that decision step. A generic action token consists of:

- **Action type (1-hot vector):** Encodes the category of the decision made (4-dimensional).
- **Tile selection (1-hot vector):** For decisions requiring the selection of a tile (Growth, Movement) (55-dimensional).
- **Unit selection (multi-hot vector):** For decisions involving the selection of one or more units from a tile (Movement, Overpopulation), this vector has length equal to the maximum stack size $S_{\max}$ and encodes which units are selected (6-dimensional).
- **Battle selection (1-hot vector):** For Battle decisions, this vector selects which unresolved battle is being resolved, worst case scenario a battle is ocurring in every tile by one single nation (55-dimensional).

All components of an action token are concatenated into a single structured representation corresponding to a single decision step. At each step, invalid entries are **masked** according to the current game state and the sequence of previously selected decisions, ensuring that only legal choices can be sampled by the policy. In total, each action token is a **120-dimensional** vector. A full nation's turn is therefore represented as a sequence of action tokens rather than as a single action, naturally reflecting the sequential and conditional structure of decision-making in Hispania.

4.4 Policy and Value Head Architecture

Policy Network After the game state has been encoded by the game-state encoder E_g and the action space has been defined in terms of structured action tokens, the policy network is responsible for generating a nation's turn as an ordered sequence of such action tokens. Rather than selecting all decisions of a phase simultaneously, the policy models decision-making as an autoregressive process. At each decision step, the agent selects a single action token, with each choice conditioned on both the encoded game state and the sequence of action tokens previously generated within the same phase.

This process is implemented using a Transformer decoder, following the standard decoder architecture illustrated in Figure 4. At decision step i, the decoder receives as input the sequence of previously selected action tokens

$$A_i = [a_1, a_2, \ldots, a_{i-1}].$$

These tokens are embedded into a common latent space and first processed by a masked multi-head self-attention block. This block allows each token to attend only to previous actions, capturing dependencies between past decisions while preserving the autoregressive structure.

Next, in the second attention block of each decoder layer—the cross-attention block—the action-token representations produced by the masked self-attention act as queries, while the per-tile embeddings from the game-state encoder E_g provide the keys and values. This is exactly where the output of E_g is incorporated into the decoder, allowing each decision step to attend to the full board state—including

tile properties, unit distributions, and global features—without introducing the game-state information directly into the autoregressive action-token sequence.

During training, full decision sequences collected from self-play are provided to the decoder in parallel, with causal masking preventing access to future decisions. During inference, decisions are generated sequentially, appending each sampled action token to the input of the decoder until the phase terminates.

At each step, the decoder produces a latent representation h_i. This latent vector is first passed through an **action type head**, which applies a softmax over the possible phases (Growth, Movement, Battle, Overpopulation) to determine the category of the next action. Depending on the selected action type, h_i is then routed to the corresponding phase-specific policy heads, each of which applies independent softmax layers over the relevant fields:

- **Tile Head:** Selects the tile relevant for the action (e.g., placement, origin, or destination). Invalid tiles are masked according to game rules.
- **Unit Head:** Selects units on the chosen tile. For actions involving multiple units (Movement, Overpopulation), the Top-K probabilities from the softmax are taken to form a multi-hot vector.
- **Battle Head:** Selects which unresolved battle to resolve. Invalid battles are masked according to the current game state.

The resulting probability distributions are sampled autoregressively to generate the next action token a_i, which is then appended to the sequence for subsequent decoder steps.

Value Head The value head produces a scalar estimate of the value of the current game state from the perspective of the player to move. It operates directly on the per-tile embeddings output by the game-state encoder E_g. These embeddings are aggregated by mean pooling across all board tiles to obtain a single fixed-size representation of the state. This pooled representation is then passed through a two-layer multilayer perceptron with ReLU activations to produce a single scalar output.

4.5 Training

Due to the large state and action space and the absence of pre-existing human game data, training will begin on a smaller, simplified map. This reduces the combinatorial complexity of the game and facilitates effective exploration of strategies. Once the agent demonstrates competence on the simplified map, training will be extended to larger maps and more complex scenarios.

Training Loop. The proposed training loop consists of the following steps:

1. The agent plays self-play games on the simplified map, recording trajectories of encoded game states, phase-specific actions, and resulting rewards.
2. The policy network is updated to maximize expected return, and the value network is trained to predict long-term rewards using standard reinforcement learning algorithms.
3. After a set number of iterations, the agent can be evaluated on progressively larger maps to assess generalization and determine the number of training iterations required for robust performance.

5 Evaluation

Since no prior artificial intelligence systems for **Hispania** are documented in the literature, the evaluation of this project must rely on a custom benchmarking environment. The assessment will follow a staged methodology designed to measure whether the proposed model is learning meaningful strategies and whether it can ultimately approach or surpass the capabilities of the existing Hispania AI.

First, the system will be tested against a baseline agent performing random or naive actions. This verifies that the model is learning coherent policies, improving over trivial play, and correctly interpreting the game-state representation.

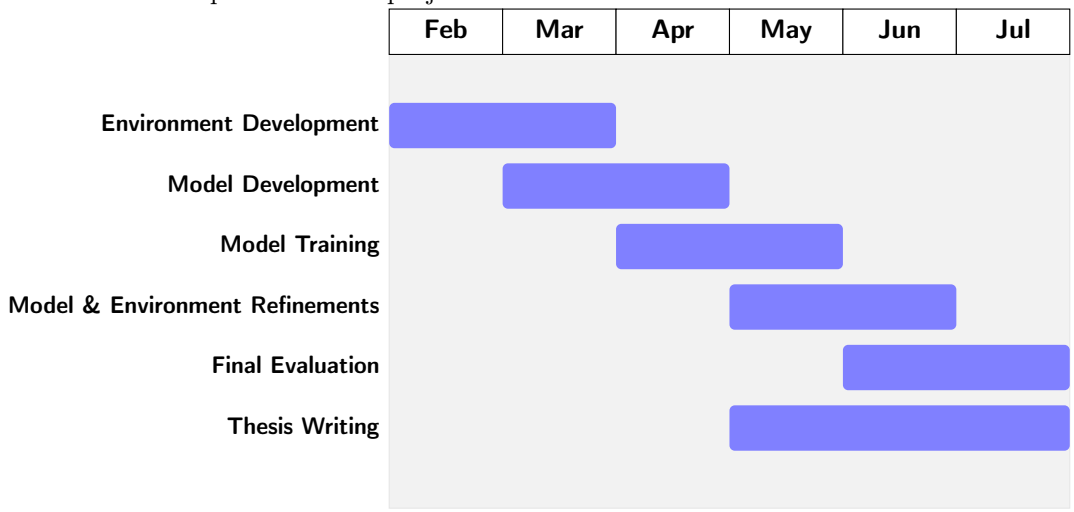
Next, the AI will be evaluated across progressively more complex self-play scenarios, including games against earlier versions of the model. This helps assess stability, learning progression, and strategic consistency.

Finally, the model will be compared directly with the current Hispania AI, which acts as the upper benchmark for this project. Multiple matches will be executed under controlled conditions to evaluate win rates, strategic depth, and the ability of the learned policies to handle the full-scale dynamics of the game.

Throughout all experiments, several performance metrics will be collected, including win rate, average decision time and the degree to which learned behaviors align with historically incentivised actions embedded in the game design.

6 Work Schedule

The work schedule expected for this project:



7 Conclusion

In conclusion, this project aims to develop an artificial intelligence agent capable of learning to play complex strategy wargames with large state spaces, structured action spaces, unit stacking, and imperfect information. Hispania is used as the primary case study due to its strategic depth and representative mechanics. While the system is designed and evaluated within the context of Hispania, the proposed model architecture and training approach are intended to be adaptable to other wargames with similar characteristics.

By combining transformer-based state representations with structured, phase-dependent action modeling and self-play training, this work provides a concrete framework for studying reinforcement learning in complex strategic environments. The resulting system aims to establish a foundation for future research on scalable and generalizable reinforcement learning methods for wargames and related strategy games.

Bibliography

[1] A. Bakhtin, D. J. Wu, A. Lerer, J. Gray, A. P. Jacob, G. Farina, A. H. Miller, and N. Brown, "Mastering the Game of No-Press Diplomacy via Human-Regularized Reinforcement Learning and Planning," in *International Conference on Learning Representations (ICLR)*, 2023.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go Through Self-Play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.

[4] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

[5] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte Carlo tree search: A review of recent modifications and applications," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 2497–2562, 2023.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[7] T. Bonjour, M. Haliem, A. Alsalem, S. Thomas, H. Li, V. Aggarwal, M. Kejriwal, and B. Bhargava, "Decision Making in Monopoly using a Hybrid Deep Reinforcement Learning Approach," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 6, pp. 1335–1344, 2022.

[8] Q. Gendre and T. Kaneko, "Playing Catan with Cross-dimensional Neural Network." Springer International Publishing, 2020, pp. 580–592.

[9] O. Vinyals, I. Babuschkin, W. M. Czarnecki *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, 2019.

[10] J. Gray, A. Lerer, A. Bakhtin, and N. Brown, "Human-level Performance in No-Press Diplomacy via Equilibrium Search," in *International Conference on Learning Representations (ICLR)*, 2021.

[11] A. P. Jacob, D. J. Wu, G. Farina, A. Lerer, H. Hu, A. Bakhtin, J. Andreas, and N. Brown, "Modeling strong and human-like gameplay with KL-regularized search," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9695–9728.

[12] H. Charlesworth, "Learning to play Settlers of Catan with Deep RL," https://settlers-rl.github.io/, 2022.

[13] G. Palma, P. A. Santos, and J. Dias, "Playing Hex and Counter Wargames using Reinforcement Learning and Recurrent Neural Networks," *arXiv preprint arXiv:2502.13918*, 2025.

The official rulebook for the board game is available at https://gamers-hq.de/media/pdf/f7/08/8c/rulebook-HISPANIA-ENG_00001-comprimido.pdf.

The digital version of the game can be found on Steam at https://store.steampowered.com/app/1963420/Hispania/.

Figure 14 shows an example of the game board as presented in the digital version of *Hispania*.

Figure 14: Digital game board of *Hispania*.