Exploring Learning Classifier System Behaviors in Multi-action, Turn-based Wargames

Garth J.S. Terlizzi III

EXPLORING LEARNING CLASSIFIER SYSTEM BEHAVIORS IN MULTI-ACTION, TURN-BASED WARGAMES

THESIS

Garth J.S. Terlizzi III, Second Lieutenant, USAF

AFIT-ENG-MS-22-M-066

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-22-M-066

EXPLORING LEARNING CLASSIFIER SYSTEM BEHAVIORS IN MULTI-ACTION, TURN-BASED WARGAMES

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Garth J.S. Terlizzi III, B.S.C.S.

Second Lieutenant, USAF

March 24, 2022

AFIT-ENG-MS-22-M-066

EXPLORING LEARNING CLASSIFIER SYSTEM BEHAVIORS IN MULTI-ACTION, TURN-BASED WARGAMES

THESIS

Garth J.S. Terlizzi III, B.S.C.S.
Second Lieutenant, USAF

Committee Membership:

Lt Col David W. King, PhD
Chair

Gilbert L. Peterson, PhD
Member

Douglas D. Hodson, PhD
Member

AFIT-ENG-MS-22-M-066

Abstract

State of the art game-playing Artificial Intelligence research focuses heavily on non-symbolic learning methods. These methods offer little explainable insight into their decision-making processes. Learning Classifier Systems (LCSs) provide an alternative. LCSs use rule-based learning, guided by a Genetic Algorithm (GA), to produce a human-readable rule-set. The learned rule-set may be given to a human player, upon which the player can perform exactly as the agent would. This thesis explores LCS usefulness in game-playing agents for multi-agent wargames. Several Multi-Agent Learning Classifier System (MALCS) variants are implemented in the wargame Stratagem MIST: a Zeroeth-Level Classifier System (ZCS), an eXtended Classifier System (XCS), and an Adaptive Pittsburgh Classifier System (APCS). These algorithms were tested against baseline agents as well as the Online Evolutionary Planning (OEP) algorithm. In a round-robin comparison among the agents, all LCS agents outperformed the baselines and OEP. APCS is the most effective game-playing agent while producing the most explainable output. ZCS and XCS outperformed the baselines and produced interpretable rule-sets. The results highlight the ability for symbolic methods to learn a complex wargame, outperform non-symbolic competitors, and provide replicable instructions.

Acknowledgements

I would like to express my appreciation to my faculty advisor, Lt Col David King, for his guidance and support during this thesis effort. His wealth of knowledge and willingness to provide help at every corner was invaluable to my academic pursuits. I also would like to thank my fellow students (you know who you are) for making this assignment the best it could possibly be. Finally, I would also like to thank my wife for the encouragement and support provided during my time at AFIT. My thesis would be nothing near the finished product if it was not for her understanding and sympathy of this thesis's challenges.

Garth J.S. Terlizzi III

Table of Contents

Page

List of Figures

List of Tables

EXPLORING LEARNING CLASSIFIER SYSTEM BEHAVIORS IN MULTI-ACTION, TURN-BASED WARGAMES

I. Introduction

1.1 Problem Background

Since the inception of Artificial Intelligence (AI), complex games provide a challenging domain for testing and exploring search techniques. Even Alan Turing proposed the game of Chess as an AI benchmark, proposing a game-playing algorithm that he would execute by-hand [1]. The importance of games in search algorithms was later expanded upon in 1958, when Arthur Samuel created a checkers-playing program using an early version of reinforcement learning [2]. Since the 1970s, the focus was on chess-playing competitions that eventually paved the way for IBM's Deep Blue program to defeat world Chess champion Gary Kasparov in 1996 [3]. In more recent years, game-playing AI is still at the forefront of the field. In 2016, Google DeepMind's agent for the classic large state-space game Go beat a world-champion in competition [4], marking a milestone in AI development.

Games have large search spaces that are difficult to fully explore. Humans are exceptionally good at developing strategies for complex games, despite the number of states and combination of actions one could take. However, with advancements in computational power and algorithmic theory, AI agents are now able to beat world-champion humans at complex games. These modern game-playing algorithms often escape human understandability and seek to replace human decision-making instead of augmenting it.

There exists general efforts throughout the AI field to make AI more transparent, explicable, and interpretable [5]. Yet, within the game-playing AI field, this effort is sparse with learning agents, as modern agents traditionally use non-symbolic methods. A symbolic agent represents its knowledge by using symbolic descriptions of the learned concepts, while a non-symbolic agent represents its knowledge in internal formats such as weighted synapses, logic units, or networks of connections [6]. Games could especially benefit from the explainability of symbolic methods to help human users understand the game and improve their performance when the AI's assisted reasoning is removed [7].

One example of symbolic AI for game-playing is the family of Learning Classifier System (LCS) algorithms. LCSs are rule-based learning machines that employ a Genetic Algorithm (GA) to discover new rules [8]. LCS implementations are divided into two families known as Michigan and Pittsburgh-style. Michigan-style LCSs evaluate the fitness of individual rules while Pittsburgh-style LCSs evaluate the fitness of rule-sets. Recent research efforts overwhelmingly focused on Michigan-style implementations [8]. Furthermore, Michigan-style LCS implementations are the widely preferred system over their Pittsburgh-style counterpart in games due to smaller evaluation times and online-learning ability. However, they contain lower reasoning ability due to the large number of rules [7]. While Pittsburgh-style LCSs often possess limitations in real-time strategy games [9], they otherwise show promise in other game settings where offline learning is possible [10] while maintaining explainability [11].

This thesis explores the application of LCS for a specific class of games: wargames. Wargames do not have a standard definition. The 2020 version of the Joint Publication 5-0 defines a wargame as "representations of conflict or competition in a synthetic environment, in which people make decisions and respond to the consequences of those decisions." [12] To contrast, the Defense Modeling & Simulation Coordination Office (DMSCO) uses the now defunct Institute of Electrical and Electronics Engineers (IEEE) 610.3-1989 definition for wargame, defining it as "a simulation game in which participants seek to achieve a specified military objective given pre-established resources and constraints; for example, a simulation in which participants make battlefield decisions and a computer determines the results of those decisions." [13] For the purposes of this thesis, we define a wargame as a battlefield simulation modeled after real-world logic in which participants make decisions to accomplish one or more objectives, and a computer determines the results and interactions.

This thesis focuses on LCS implementations in the turn-based, simultaneous-move, and multi-action game Stratagem MIST. Stratagem MIST is a game in-development by the Air Force Research Laboratory (AFRL) and serves as a multi-domain simulator for wartime conflicts. To the best of the author's knowledge, an LCS agent dedicated to a military wargame is currently unexplored. Furthermore, the lack of research involving Pittsurgh-style LCSs in games in general provides a point of interest and unexplored territory. Our hypothesis is that LCSs, specifically Pittsburgh-style LCSs, can be an effective symbolic game-playing agent for Stratagem MIST in addition to producing explainable outputs that can allow an outside observer to understand its decision-making process.

1.2 Motivation

Wargames are especially important to the field of military science, as they can be used for development of combat theory [14]. The concept of the modern wargame was invented in Prussia in 1780, where records indicate young officers played a table-top battlefield game to learn military strategy. Historically, lessons learned from wargames often translate to real-world scenarios. In World War I, every major combatant employed wargaming to aid in war planning [15]. In World War II, the United States and British Royal Navy used the outcome of analyzed wargames to develop better tactics while axis forces used wargames to predict allied strategies [14]. In recent years, independent projects demonstrated the success of applying AI techniques to solve complex wargames [16]. However, the goal of solving games or developing advanced search techniques to outperform humans may conflict with the established purpose of wargames. If the objective of a wargame is to prepare a human player for actual conflict through simulation, the actions of a non-symbolic agent may not transfer into real-world representation. In contrast, an agent that can play a wargame well and present its rationale in an interpretable form could potentially help discover new strategies and tactics that can be translated to real-world scenarios.

Stratagem MIST is a prime domain to explore the use of LCSs in wargames. Its large complexity, general wargame structure, and adaptability to multiple multi-domain scenarios prompt research interest on multiple fronts. As other LCS implementations are absent from wargames and Stratagem MIST, it is important to track the internal makeup of LCS rules with regard to condition distribution, action distribution, and action selection tendencies. This data may answer questions regarding LCSs' ability to make complex decisions while maintaining explainability and performance.

1.3 Research Questions

This thesis seeks to answer the following research questions:

1. How effective are LCS agents in a wargame setting?

2. What does the internal makeup of the LCS-generated rules look like for Stratagem MIST?

3. How interpretable are the default rule-sets generated by LCS agents, and what can be done to improve interpretability?

1.4 Research Tasks

The following tasks were performed to answer the aforementioned research questions:

1. Develop agents that can effectively play Stratagem MIST using known LCS methods.
2. Conduct experiments comparing the win rate of agents using different symbolic and non-symbolic game-playing methods.
3. Analyze how well the LCS agents can develop strategies to general Stratagem MIST scenarios.
4. Examine the distribution of conditions and actions in finalized LCS rule-sets.
5. Analyze how the resulting rule-sets can be interpreted and reduced to a readable form.

1.5 Document Overview

Chapter II presents an introduction to game theory concepts, an overview of LCS research, and a description of Stratagem MIST. Chapter III outlines the efforts implemented to answer the research questions. Chapter IV analyzes and discusses the resulting data. Finally, Chapter V presents conclusions and discusses future work.

II. Background and Literature Review

Game-playing agents that make complex decisions continue to be at the forefront of the field of Artificial Intelligence (AI). While the field witnessed a recent shift away from explainable AI [17], a return to the Learning Classifier System (LCS) family of algorithms provides a multi-objective approach to develop effective game-playing agents while maintaining explainability. In this chapter, Section 2.1 describes the core components of decision theory and game theory. Section 2.2 covers genetic algorithms, which are a crucial part of LCSs. Sections 2.3 and 2.4 outline the architecture of various LCS implementations and their history in games. Finally, Section 2.5 describes the domain of Stratagem MIST.

2.1 Decision Theory

A familiarization of how agents make decisions is required to understand the nuances of symbolic game-playing methods. The concept of developing agents that can perform complex tasks in games falls under the broader scope of decision theory. Decision theory encompasses the notion of attempting to achieve a maximum expected utility by combining probability knowledge with a utility function, and a problem that can be solved using elements of decision theory is known to be a decision problem [18].

2.1.1 Markov Decision Process

Some of the algorithms described in this thesis use a Markov Decision Process (MDP) to make their decisions. A MDP [18] is a sequential decision problem for a fully observable and stochastic environment, often used in reinforcement learning, with the following components:

- s is a set of states (with an initial state s_0).
- a is a set of actions.
- $P(s'|s, a)$ is a Markovian transition model. A transition is Markovian if the probability of reaching s' from s is independent of the history of previous states.
- $R(s)$ is a reward function.
- π is a policy, a solution that specifies what an agent should do for any state the agent may reach. $\pi(s)$ is the action recommended by the policy at state s.
- π^* is the policy that creates the optimal utility.

2.1.2 Decision Trees

A decision tree [18] represents a function that takes in a vector of inputs and returns a single output, known as a decision. This decision is a made by performing a sequence of tests. All decision trees start at the root and and traverses conditions until a decision is formed. This document will explore a specific type of decision tree, the Fast And Frugal Tree (FFT) [19]. A FFT is a specific type of binary decision tree that has a single condition as a root and each level of the tree contains a single action if the previous condition was met, or a branching condition otherwise. The only exception is the deepest level of the tree, which has a default action. A diagram of a FFT is shown in Figure 1. A FFT is also known as a decision list [20], and its usefulness is derived from its interpretability and learnability.

2.1.3 Game Theory

Game theory is a direct extension of decision theory that focuses on games, a decision simulation which can be described formally as [21]:

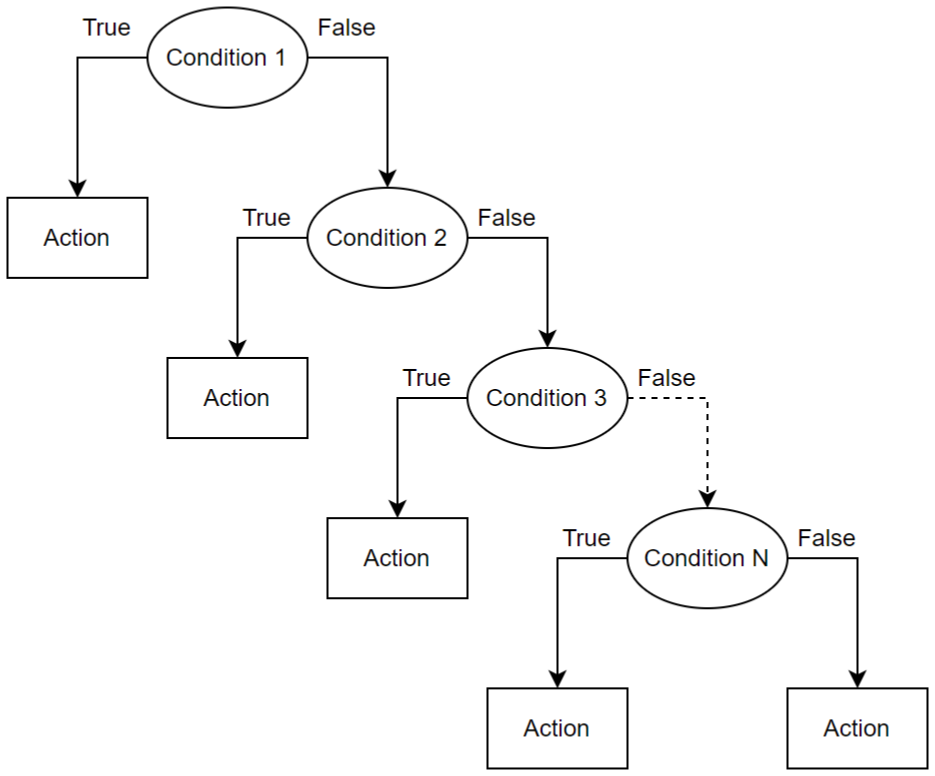


Figure 1: A Fast and Frugal Tree of Depth N [19].

- S is a set of states (with an initial state s_0).
- S_t is a set of terminal states in s that upon reaching, the game ends.
- N is the number of players.
- A is the set of actions.
- $f(S, A)$ is the state transition function that returns a new state.
- $R(S)$ is the reward function associated with a specific state.

Within this definition of a game, there are many attributes that make up game theory, that can define a game and impact its complexity. These attributes are aggregated from [18], [21], and [22]:

- Zero-sum/Non-Zero Sum: A game is considered zero-sum if the sum of the payoffs for each action is equal to zero. In other words, a move that benefits a player will equally detriment other players. In a non-zero sum game, an action that is beneficial to a single player will not necessarily adversely impact other players.

- Cooperative/Non-cooperative: A game is considered cooperative if there exists terminal states desirable to multiple players, such that multiple players can win the game. Otherwise, the game is considered non-cooperative.

- Symmetric/Asymmetric: A game is considered symmetric if all players start off with the same assets. At the start of the game, all players should possess an equal chance of winning. In an asymmetric game, players start off with different assets or available actions.

- Sequential/Simultaneous: A game is considered sequential if the state transition function is only dependent on the action of one player. Turn-based games are considered sequential. A game is considered simultaneous if the state transition function takes into account the actions of multiple players. The actions do not need to be selected by the player at the same time, but can be queued for simultaneous execution.

- Perfect Information/Imperfect Information: A game is considered having perfect information if all aspects of the game's state are visible to the player. For example, chess is a perfect-information game, as all pieces and potential actions are able to be viewed by both players at all times. A game of imperfect information will have some component of the game's state hidden from the player. For example, the poker variant Texas Hold 'em involves the players holding cards in secret from each other [19].

- Discrete/Continuous: A game is discrete if it has a finite set of states, actions, and players. Otherwise, the game is continuous.
- One-Player/Many-Player: A game is considered one-player if actions are chosen by a single player. It is important to note that a game played by a single human can still be considered many-player if actions are being chosen by artificial agents.

The game being analyzed in this document, Stratagem MIST, is a zero-sum, non-cooperative, simultaneous, perfect information, discrete, many-player game. A specific game's classification as symmetric or asymmetric is dependent on the game mode and specific scenario.

2.2 Genetic Algorithms

All non-baseline agents explored in this document use a Genetic Algorithm (GA) in their learning process. A GA is a specific type of evolutionary algorithm that possess populations of chromosomes, parent selection according to fitness, crossover to produce new offspring, and random mutation of new offspring [23]. The psuedocode for a general GA can be found in Algorithm 1, and the individual components described in [23] are:

- Populations of chromosomes - A chromosome can be viewed as an entity of changeable genes that makes up the individual. The initial population should include diversity in chromosomes.
- Selection According to Fitness - The process of selecting chromosomes in the population for reproduction. Typically, a chromosome's probability of selection is proportional to its fitness function, which is a metric defining the performance of an individual.

- Crossover - The process of combining parent individuals to create children offspring with chromosomes derived from a combination of both parents. The crossover process is derived from the biological recombination process between two single-chromosome (haploid) organisms.
- Mutation - The process of randomly altering a chromosome in an attempt to create genetic diversity.

Algorithm 1 Genetic Algorithm

```
    numPop = # of Individuals in Population
    numIt = # of Iterations
    pMut = probability of Mutation
 1: function GA($numPop, numIt, pMut$)
 2:     Generate population of size numPop
 3:     Calculate fitness of Initial Population
 4:     for $currentIteration \leftarrow 1, numIt$ do
 5:         Select two parents from current population (often with high fitness scores).
 6:         Perform a crossover on the parents.
 7:         if $rand(0,1) < pMut$ then
 8:             Mutate the offspring
 9:         end if
10:         Calculate fitness function of each child.
11:         Add offspring to the population.
12:         Remove least-fit offspring until the current population is equal to numPop.
13:     end for
14: end function
```

2.2.1 Co-evolutionary Algorithms

Stratagem MIST is a multi-action game that requires units to work together to accomplish an objective. Thus, it is a prime domain for Multi-Agent Systems (MASs), where each unit acts as its own agent while co-evolving with other units. Potter and De Jong [24] expanded on the concept of GAs with regard to MASs, systems that use more than one agent to accomplish a learning task proposing the first concept of a Cooperative Co-evolutionary Evolutionary Algorithm (CCEA) as a method for function optimization. Whereas GAs showed success with developing fit individuals through direct competition, they are also able to cooperatively evolve a group of individuals. This concept was expanded upon in [11], where Grefenstette and Daley outlined four co-evolutionary approaches for evolving competing agents. In the first method, an agent is evaluated against a random member of the opposing population. In the second method, an agent is evaluated against the previous champion of the opposing population. In the third method, an agent is evaluated against a set of previous champions. In the fourth method, an agent is evaluated against a preliminary prediction of the champions of the current generation. Section 4.1.4 outlines the co-evolutionary methods chosen in the experiments.

2.3 Learning Classifier Systems

A notable example of the success of GAs is shown in the LCS family of algorithms. A LCS is a rule-based system that converts a state-space to a set of condition-action pairs. Over the past four decades, since the inception of the original LCS, Classifier System One (CS-1) developed by Holland in 1978 [25], LCSs have seen many iterations and implementations. While there are many different variations, Holmes *et al.* [26] define a generalized LCS as a system that has the following properties:

1. *A population component* that consists of classifiers (condition-action rules with their associated parameters) that represent the current knowledge of the system.
2. *A performance component* that moderates interaction with the environment.
3. *A reinforcement component* that determines how certain classifiers are rewarded based on their fitness.
4. *A discovery component* that seeks to discover new rules, usually through a genetic algorithm.

A recent area of interest in the field involves the extension of classic LCS algorithms to Multi-Learning Classifier Systems (MLCSs) [27]. These include Multi-Agent Learning Classifier Systems (MALCSs), ensemble classifiers, and distributed classifiers. This thesis focuses on MALCSs, MASs in which agents utilize a LCS.

There are two major types of LCS implementations which differ based on their representation of the solution [8]: Michigan and Pittsburgh-style. The Michigan-style LCS encompasses all LCS implementations modeled after University of Michigan researcher Holland's breakthrough 1978 proposal of LCS [25]. In a Michigan-style LCS, a classifier consists of a single condition-action rule and its parameters. Each classifier is evaluated individually, such that rules can easily be added, mutated, and removed without severely altering the functionality. The Pittsburgh-style LCS encompasses all algorithms that are modeled after Smith's 1980 initial algorithm [28] at the University of Pittsburgh. In a Pittsburgh-style LCS, the classifier consists of a set of rules which are evaluated collectively. The complete solution is represented by the complete rule-set. Both LCS styles differ fundamentally in how the agent formulates its classifiers and learns from the environment.

2.3.1 Michigan-Style LCSs

Because the term Michigan-style LCS is not a system but a family of systems, it is difficult to create a unified architecture for it. However, Urbanowicz and Moore's generic LCS model [8] inspired by the most popular implementations of LCSs encapsulate the core of many Michigan-style algorithms. The architecture can be viewed in Figure 2, and the individual components are explained in the rest of the section, with the exception of Section 2.3.1.9.

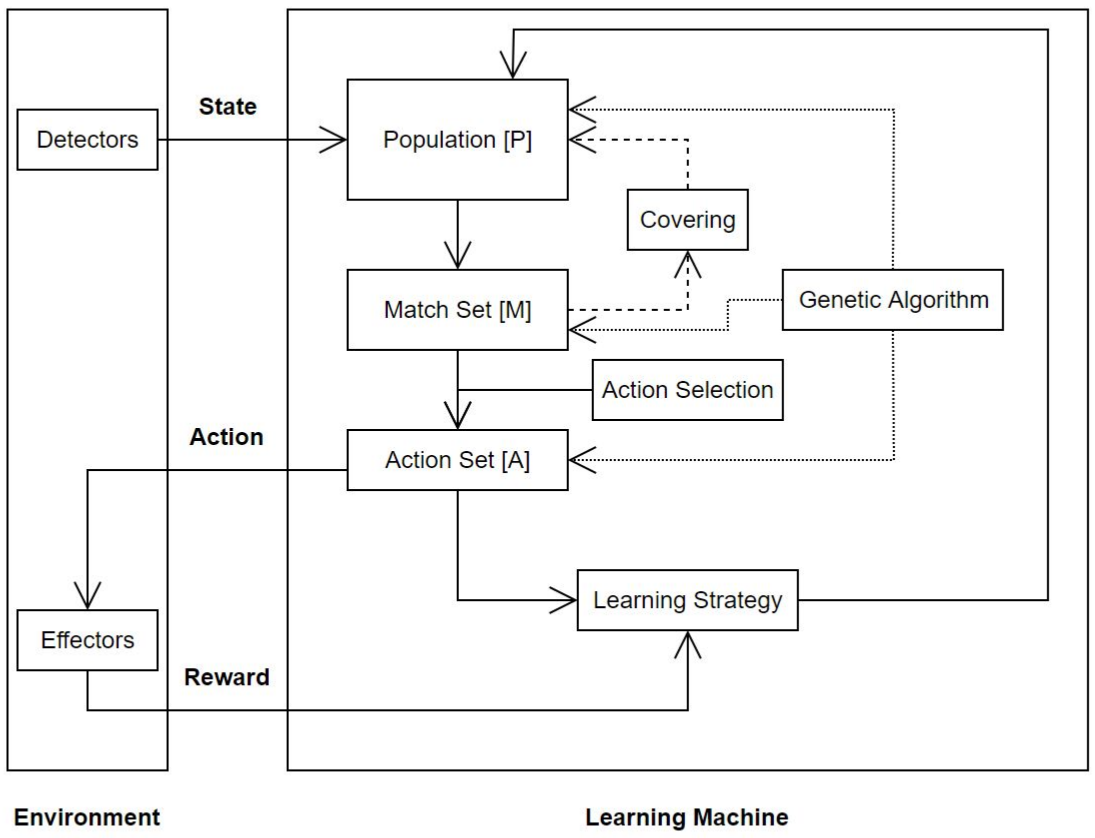


Figure 2: A Generic Michigan-style LCS that encompasses elements from many Michigan-style implementations [8].

2.3.1.1 Interaction with the Environment

The first step of the LCS process is interaction with the environment, through mechanisms known as detectors and effectors. Detectors sense the environment and parse an environmental instance into a usable condition-string, which is then placed into the Learning Machine (all other non-environmental components of LCS). Following the outputted action of the Learning Machine, the effector observes the action's effect on the environment and returns a reward.

2.3.1.2 Population [P]

The population set, denoted as [P], consists of classifiers. The classifiers in [P] can be either initially randomly generated or empty, and filled by the process known as covering. The classifiers in [P] are allowed to contradict each other, such that two classifiers with the exact same condition string lead to different actions. Throughout the LCS process, the classifiers in [P] are added, mutated, or removed. Most LCS implementations also have some form of population controller that deletes ill-formed conditions that rarely match an environmental instance, or consumes a specific rule into a more general rule through a process known as subsumption.

2.3.1.3 Covering

Covering is a mechanism to introduce new classifiers into [P]. Generally, it is only applied if no classifiers in the population match the environment instance. The primary goal of covering is to verify that there is at least one classifier in [P] that matches the current training instance. Through covering, a rule is randomly generated that will match the training instance, and an action is randomly assigned. The rule should be general enough to match similar environment instances, but not specific enough where it will match infrequently.

2.3.1.4 Match Set [M]

The Match Set, denoted as [M], is the subset of classifiers in [P] that matched the rule of the environment instance. If a classifier was generated due to covering, it is automatically included in [M] following the initialization.

2.3.1.5 Action Selection

From the classifiers in [M], the LCS must choose an appropriate action. A classifiers' parameters are often the determining factor in this decision. A classifier often has a fitness score that determines its value among other classifiers. This fitness is often strength-based or accuracy-based. Under a strength-based fitness, classifiers are evaluated by the expected reward to be received if the classifier's action is utilized. Accuracy-based fitness measures a classifier's accuracy, such that the system develops a more complete map of the problem space and avoids focusing on only higher-payoff niches.

Action Selection components also often have a type of exploration vs. exploitation mechanism, such that the system prioritizes high-payoff classifiers but do not get stuck in a local maximum. Thus, many action selection systems will have the option to randomly choose an action from [M] even if its fitness is lower. Other common action selection policies include roulette selection based on fitness, or tournament selection with a sample of the population.

If the task of the classifier is a reinforcement learning task, a prediction array is often used. The prediction array is a list of prediction values calculated for each action found in [M]. In certain reinforcement-based LCS implementations, the prediction array can have a form of credit assignment that predicts the Q-value of a particular action.

2.3.1.6 Action Set [A]

The Action Set, denoted as [A], is the subset of classifiers in [M] whose action corresponds to the one chosen by the action selection component. These actions are channeled back to the environment, upon which the effectors in the environment determine the reward for those actions.

2.3.1.7 Learning Strategy

The Learning Strategy is often composed of a fitness evaluation phase followed by a parameter update phase. The learning strategy is implementation-specific, and often is the component that differentiates two LCS algorithms from one another.

2.3.1.8 Genetic Algorithm

Depending on the specific LCS implementation, the Genetic Algorithm can be implemented with the classifiers in [P], [M], or [A]. The associated fitness value of each classifier is often the determining factor behind selection. In a Michigan-style LCS, the GA performs at the rule-level, as each chromosome is represented as a set of conditions followed by a proposed action. Mutation occurs by altering a single condition or action, and crossover occurs by merging two condition sets and selecting a single action among two parent classifiers. LCSs often have a deletion mechanism to detect defunct classifiers generated by a GA. A detailed description of a general Genetic Algorithm not specific to LCS can be found in Section 2.2.

2.3.1.9 Popular Michigan-Style Algorithms

Many modern Michigan-style algorithms stem from the simple architecture found in Wilson's Zeroeth-Level Classifier System (ZCS) and eXtended Classifier System (XCS) algorithms [8]. ZCS [29] is a strength-based extension of Holland's initial LCS. The algorithm's contribution to the field can be attributed to the use of a credit assignment scheme in its learning strategy. The original algorithm uses a combination of Q-Learning and Holland's Bucket Brigade strategy, a credit assignment scheme that mirrors an information-based service economy where classifiers must "bid" a portion of their strength to receive a future reward [30]. ZCS simplifies the CS-1 algorithm it was derived from, removing message-passing and heuristics to improve understanding of the algorithm.

XCS [31] is an extension of ZCS that altered the field by evaluating a classifier's fitness based off the accuracy of the rule as opposed to the strength. While many variations and extensions exist, an XCS is generally characterized by an accuracy-based fitness score, niche-based rule discovery occurring in the action set, and alternating exploration and exploitation trials [32].

2.3.2 Pittsburgh-style LCS

Shortly after Holland proposed what would later be referred to as a Michigan-style LCS, researchers at University of Pittsburgh, led by Smith, established what would later be referred to as the Pittsburgh-style LCS. Whereas a Michigan-style LCS can be viewed as an iterative, continuous convergence of one solution, a Pittsburgh-style LCS can be viewed as a generalization optimization process [33]. Because Pittsburgh-style LCSs evaluate whole rule-sets, they often are not scalable, and are often limited in size. Thus, the implementation of a Pittsburgh-style LCS involves a trade-off: the rule-set must find a balance of minimalism and accuracy [34]. A generic Pittsburgh-style LCS is shown in Figure 3.

There are two common Pittsburgh-style algorithms that are often applied to learning tasks: GALE [35] and GAssist [36]. Urbanowicz and Moore [8] note the most successful Pittsburgh-style implementations are designed for data mining and classification problems, and are limited by their ability to only learn offline.

The SAMUEL algorithm [11] is a common implementation of Pittsburgh-style learning in game domains. The algorithm uses a population of randomly-generated decision lists consisting of high-level rules to select actions based on signals from the environment. As a SAMUEL agent progresses throughout the environment in an episode, it iterates sequentially through its decision list to select the action corre-

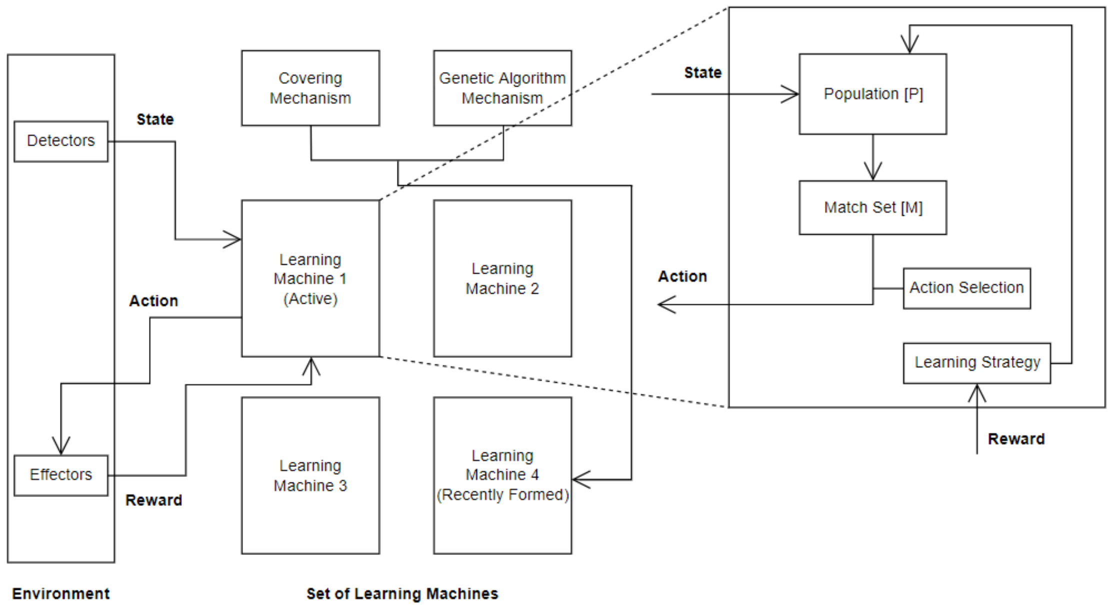


Figure 3: A Generic LCS that encompasses elements from many Pittsburgh-style implementations.

sponding to the first matched rule. After k episodes conclude, a cumulative reward is assigned to the decision list to be used as its fitness. After each individual in the population has received a fitness, a GA preserves the fittest in the population and forms a new generation through crossover and mutation.

The Adaptive Pittsburgh Classifier System (APCS) algorithm [37] is an adaptation of Smith's algorithm designed for reinforcement-learning problems that develops a population of fixed-sized classifiers. Like SAMUEL, it uses a population of rules to make decisions, and then applies a GA to form new rule-sets. However, there are a few distinctions from SAMUEL. First, rules do not need to be represented in symbolic format. Secondly, online adaptive covering may be applied if no rule in the rule-set matches the environment. Finally, there is no fixed action selection policy. While SAMUEL's decision-list/FFT action-selection policy may be used, other policies include selecting the most general matched classifier, the most specific matched classifier, or a random matched classifier.

2.3.3 Advantages of LCSs

The main advantages of LCSs are derived from its interpretability in form of the use of symbolic methods [7]. LCS implementations apply a form of symbolic reasoning to complete their learning tasks. This phenomenon can outperform other tasks when the objective is to mimic human thinking or provide a platform for human understanding. A human cannot be expected to understand the agent's decision-making process given the end product (a set of weights) of a trained neural network, but can traverse symbolic rules and come to understand the reasoning behind a LCS's decision-making process.

Furthermore, because LCS implementations provide instructions for how to handle generalized components of a state-space as opposed to the complete state-space, LCSs are not as computationally expensive as other traditional game-playing methods. This phenomenon was observed in [38], where the learning rate parameter for the Michigan-style LCS variant XCS was significantly higher than an agent that used the Neuroevolution of Augmenting Topologies (NEAT) algorithm.

Due to the evolutionary-based search of the discovery component, LCS implementations are competitive in domains where error correction cannot be directly determined [34]. Applying their interactive evaluation-evolution approach, LCS credit assignment policies use responses from the environment but they do not convert it directly into structural search biases. As a result, LCS implementations tend to find globally-optimal solutions in challenging problems, where other game-playing agents using traditional search-based approaches may get stuck in a local optima [34].

Lastly, LCSs are powerful learning tools that can outperform non-symbolic methods in some domains. For example, Urbanowicz and Moore in [39] notes that other machine learning algorithms have trouble adapting to scaled, distributed, and non-linear problems, citing the 135-bit multiplexer as a problem only being able to be solved by LCSs. The 135-bit multiplexer is a benchmark learning task to predict the value of one of 128 register bits pointed by the first seven bits of the sequence.

2.3.4 Limitations of LCSs

LCSs suffer from a few limitations, described in [40], which may contribute to its relative sparsity to the field compared to other learning methods. First of all, it is not-widely known compared to traditional learning systems, causing fairly limited software availability and interfacing. Furthermore, it can suffer from over-fitting, even with rule compaction techniques. There are also many parameters to optimize, with un-tuned parameters such as learning rate significantly impacting performance. Moreover, it is sometimes challenging to represent domain-specific rules and extract knowledge from a state-space.

There also exists relatively little theoretical work for mapping problem domains to specific implementations. In other words, it is often difficult to take a problem and determine which LCS algorithm or hyper-parameters would be appropriate, based off existing implementations. Further research is required to understand the inner workings of applying LCSs to problem domains and what key properties are crucial for selecting a learning algorithm [34]. Thus, when applying a LCS to any problem domain, multiple design choice variations should be considered and tested. Prior to testing their effectiveness in a tournament, Stratagem MIST's LCS agents underwent preliminary experiments determining the best action selection and standardization strategies (see Section 4.1).

2.4 LCSs in Games

Despite LCSs being in existence for over four decades, their extension to games are less explored than other learning tasks [7]. While Smith proved that his initial

Pittsburgh-style LCS algorithm could be extended to a poker domain [10], the extension to real-time, multi-unit, or spatial games did not occur until much later. In 1995, Serendynski *et al.* [41] proposed what is suggested to be the first use of LCSs in a spatial game, developing Michigan-style agents on games that use both small-world and scale-free networks. Their success prompted many more game agents based off a LCS algorithm across many platforms. This list includes a ZCS for the multi-agent game Tank Battle [42], a Modular and Hierarchical Classifier System (MHiCS) for the Massively Multiplayer Online Role-Playing Game (MMORPG) Ryzom [43], and an XCS for the board game Connect4 [32].

A vast majority of modern applications of LCS in games are Michigan-style, as often performance is desired over interpretability. Due to increased complexity and domain-specific difficulties, Pittsburgh-style LCS implementations struggle in comparison to their Michigan-style counterparts in real-time scenarios [7], but are effective in some domains such as Unreal Tournament 2004 [9]. XCS, the most popular Michigan-style LCS, is shown to be an effective competitor in a wide range of games, including Othello [44], Robocode [38], and Starcraft [45].

While all LCS algorithms designated as Michigan-style must follow a version of the architecture shown in Figure 2, the term Pittsburgh-style LCS is a catch-all term for any learning processing with an evolving rule-set and a genetic algorithm. Thus, a common problem with finding its implementation in literature is that its implementation in games can often be unacknowledged by authors, perhaps inadvertently or intentionally for novelty purposes. For example, De Mesentier Silva *et al.* [19] were able to demonstrate the effectiveness in generating novice heuristics for a poker game using rule-sets as a FFT and a genetic algorithm. Gallagher and Ryan [46] were able to use an evolutionary, rule-based approach to play Pac-Man. Both implementations meet all the requirements for their systems to be considered a variant of a LCS (possessing population, performance, reinforcement, and discovery components [26]), yet the LCS concepts are absent from their papers. This phenomenon poses a potential problem to the field, as there exist many implementations that can expand the theory behind LCS in games but are unacknowledged by their authors.

There exist a few instances where MASs apply LCSs in their decision-making processes. Hercog and Fogarty [47] introduced homogeneous agents using multiple ZCS agents to co-evolve rule-sets to solve the *El Farol* problem. Notable design choices include the combination of a greedy reward scheme that benefits individual agents and a cooperative reward scheme that benefits all agents, as well as a controller that ensures all agents explore or exploit simultaneously. Hercog later used multiple XCS agents to solve the same problem in [48]. In [49], Castillo and Lurgi extended multi-agent XCS to the game Robocup. They found that the result of the performance of XCS was heavily dependent on the quality of the starting homogeneous rule-set shared by the agents.

2.5 The Game of Stratagem MIST

Stratagem MIST is a game developed by the Air Force Research Laboratory's Information Directorate (AFRL/RI). It is a graph-based, simultaneous-move, turn-based, multi-action game where units may move along a connected graph. The game can be played with multiple players and multiple teams, but the default settings consist of two teams (Red/Blue) with one player on each side. Each player is given near-perfect information. Each player may see all vertices, edges, units, and objectives. At time of writing, the game is currently in development. Each game is loaded as a scenario, which is a networked-graph map with pre-defined starting unit assets and positions. A configuration can be defined as a scenario with pre-loaded agent types, *e.g.* Red Random Agent vs. Blue Roving Mob Agent on Hex Battle. This document refers exclusively to Release Version 4.0, and thus changes made in later versions of this game will not be reflected in this document.

2.5.1 Objectives and Metrics

A game may be won via the use of objectives and/or metrics. Each team can be assigned one or more objectives. An objective is a boolean function that returns a true or false value based on the current state of the game. The current objectives are as follows:

- Kill All Enemies: The objective is met if all the enemy players have been killed.
- Capture Node: A set of nodes to capture N is given to the player. The objective is met if all nodes in N are captured by the player's team.
- Protect Unit: A set of friendly units U is given to the player. The objective is met if all units are alive at the end of a turn. If a single unit in U dies, the objective is failed.
- Territory Value Held Consecutively: A set of nodes N that form a territory, a threshold value k, and a consecutive turn limit t are given to the player. If the player holds k values in N, a turn counter is initialized to 0. For each turn where k nodes in N is captured, the turn counter is incremented. In the event that the condition is not met (the team loses control of the territory), the counter is reset to 0. The objective is met when the turn counter is equal to t.

The game may also be won via the game's system of metrics. A metric is a non-boolean scoring system that determines a player's success. The highest score wins the game. The current metrics are as follows:

- Force Ratio: This metric calculates the combined strength of all friendly units in proportion to enemy units. For example, if combined strength of the blue team is 400 and the combined strength of the red team is 600, the blue teams has a force ratio of 0.4 and the red team has a ratio of 0.6.

- Territory Controlled: This metric calculates how many nodes a player has captured relative to all available nodes. For example, in a map with 20 nodes, if the blue team has captured 5 nodes, the red team has captured 10 nodes, and 5 nodes remain uncaptured, the blue team has a territory controlled metric of 0.25 and the red team has a territory controlled metric of 0.5.

Teams may have an unbalanced set of objectives. For example, a scenario may exist where the red team's sole objective is Kill All Enemies, while the blue team's objectives are Capture Node and Protect Unit.

2.5.2 Ground Plan

Each turn, the agent must submit a Ground Plan comprised of queued move orders from different units. A move order will consist of three potential actions:

- Attack – The unit is ordered to move to an adversary-occupied node.
- Defend – The unit is not ordered to move, and instead will hold its position at its current node.
- Move – The unit is instructed to move to an unoccupied node or node with friendly units.

Units may move independently of each other, and may move towards any vertex on the graph provided there is an edge. A combat event triggers if two opposing units cross paths on an edge, or if a unit arrives at a vertex occupied by an opposing unit. Move orders carry over between turns, such that if a unit is not able to move to the designated vertex by the end of its turn, the unit will continue to travel on the next turn. Furthermore, the game uses an in-game path-finding mechanic to determine the best path between two vertices, but the user can elect to create a custom path.

A ground unit's strength is affected by the terrain it is on and whether it is attacking, defending, or moving. The movement multipliers based on terrain can be found in Table 1 and the attack/defense multipliers can be found in Table 2.

Unit Type	**Clear**	**Rough**	**Urban**
Infantry	1.0	0.8	1.0
Armored	1.2	0.5	0.25
Mech Inf	1.1	0.65	0.5

Table 1: Movement Multipliers based on Terrain

Unit Type	**Role**	**Clear**	**Rough**	**Urban**
Infantry	Attack	1.0	1.0	1.2
Infantry	Defend	1.0	1.2	3.0
Armored	Attack	1.4	0.9	0.8
Armored	Defend	1.6	1.2	1.5
Mech Inf	Attack	1.1	0.9	0.9
Mech Inf	Defend	1.3	1.2	1.8

Table 2: Attack/Defense Multipliers based on Terrain.

2.5.3 Air Plan

Each turn, the agent must submit an Air Plan consisting of four components:

1. *Air Apportionment Orders.* Orders are split up into four aircraft groups: Air Superiority, Multi-Role, Strike, and Surface-to-Surface Missile (SSM). Each group can be tasked a ratio that must be added up to one between three priorities: Defensive Counter-Air/Offensive Counter-Air (DCA/OCA), Air Interdiction, and Close Air Support (CAS).

2. *DCA/OCA Priorities.* The aircraft will prioritize a region for counter-air procedures. It applies some fraction of its air attack power against all of the other

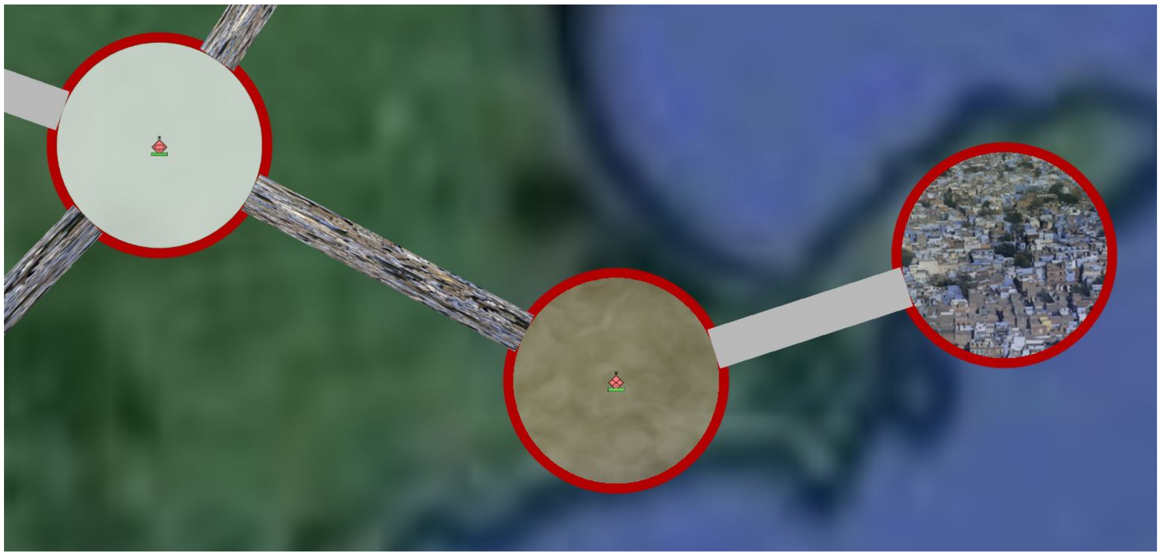


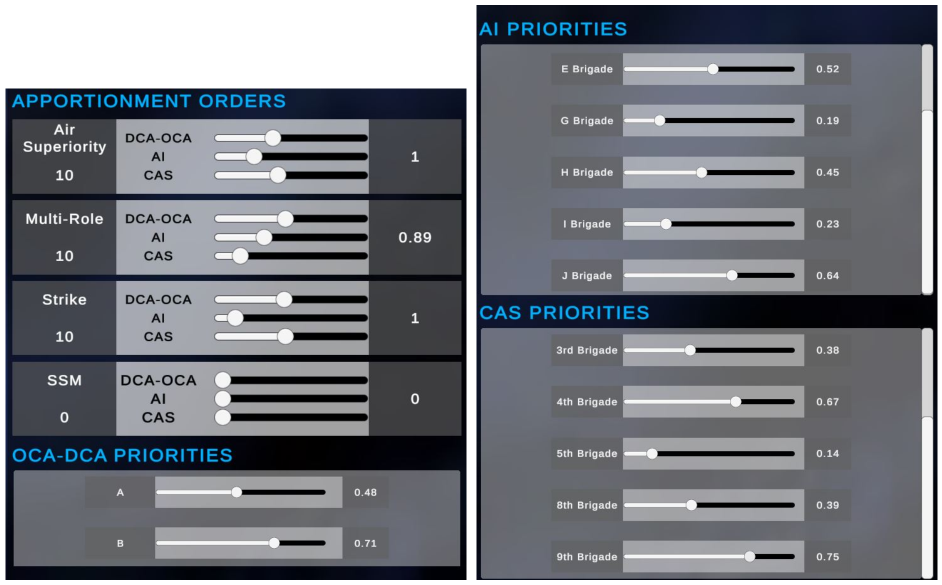
Figure 4: A subsection of a map featuring nodes with different terrains. From left to right, the terrain types of the nodes are Clear, Rough, and Urban. Edges may possess a terrain type that does not match a connected node, as shown in the two observable edges here.

aircraft that spent time in the same Air Region.

3. *Air Interdiction Priorities.* The aircraft will attack specific ground enemies. Damage dealt from the aircraft will be reflected in a targeted ground unit's strength.

4. *CAS Priorities.* The aircraft will support friendly ground units such that a bonus will be granted to ground units should they enter into a combat event with an opposing ground unit.

The in-game Graphical User Interface (GUI) menu to submit an air plan is shown in Figure 5.



APPORTIONMENT ORDERS

Air Superiority 10	DCA-OCA / AI / CAS	1
Multi-Role 10	DCA-OCA / AI / CAS	0.89
Strike 10	DCA-OCA / AI / CAS	1
SSM 0	DCA-OCA / AI / CAS	0

OCA-DCA PRIORITIES

A	0.48
B	0.71

AI PRIORITIES

E Brigade	0.52
G Brigade	0.19
H Brigade	0.45
I Brigade	0.23
J Brigade	0.64

CAS PRIORITIES

3rd Brigade	0.38
4th Brigade	0.67
5th Brigade	0.14
8th Brigade	0.39
9th Brigade	0.75

Figure 5: A sample Stratagem MIST Air Plan using the in-game menu.

III. Methodology

This chapter discusses the design and evaluation of the examined Learning Classifier System (LCS) agents, whose performance and generated rule-sets were utilized to address the questions in Chapter I. Section 3.1 lists the challenges associated with implementing agents within the Stratagem MIST environment. Section 3.2 describes the baseline agents included with the game. Section 3.3 discusses development decisions applied to all LCS algorithms. Section 3.4 outlines the learning strategies of all explored algorithms. Section 3.5 discusses Strategem MIST's environment for which the quality of the agents are tested on. Finally, Section 3.6 discusses the scoring functions used in fitness evaluation and agent evaluation.

3.1 Implementation Challenges

Stratagem MIST is a difficult domain to develop a game-playing algorithm. The game is currently in development and thus there exists no public research regarding heuristics or strategies. Furthermore, the challenges with implementing a game agent and determining the best action given a specific state in Stratagem MIST arise due to the following properties of the game:

- *The game is multi-action.* Units move independently of each other, exponentially increasing the state-space. For example, if our team has 10 units with 20 actions each, then our total action set A contains 20^{10} moves [50].
- *The game is simultaneous-move.* From any state S, if the set of our agent's available actions are defined as A_1 and the opposing agent's available actions are defined as A_2, a state-space S' is generated from any a_1, a_2 pair where $a_1 \in A_1$ and $a_2 \in A_2$. Thus, the branching factor for a given ply will be $|A_1 \times A_2|$ [51].

- *The game is multi-objective.* Because there is not a single metric to determine the winner of the game, it is difficult to determine a single fitness score. Does a state space that places a high value in a single objective have a better evaluation than a state space that has a more even score distribution across objectives? This evaluation choice makes heuristic-based algorithms difficult to implement.
- *The game's units have internal state.* Each unit contains a strength component which determines its success in a combat event. Therefore, a decision must be made in how to represent this value in a state. One implementation could choose to represent state as a pure decimal value while another implementation could separate unit strength into discrete buckets. The way this state is represented could cause significant variation across similar implementations.
- *The game's environment varies across scenarios.* Stratagem MIST is not played in a fixed environment, and each scenario has different configurations of network, units, and objectives. Thus, an agent must be able to generalize their game-playing strategy across scenarios.

3.2 Baseline Agents

Stratagem MIST Release 4 provides three baseline agents to play against:

1. Random: This agent will move all units independently of each other. All Air Plans are generated randomly.
2. Roving Mob: This agent moves all of its agents into a single node and then moves as a cohesive mob to attack the enemy. It can be paired with the following Air Plans:
    (a) Murder All Air Bases: This agent works by targeting enemy AirBaseUnits exclusively at the cost of everything else.

(b) Air To Ground Focus: This agent is designed to take all of its aircraft and set it on an Air Interdiction task. For each turn, a single ground unit will be targeted by all aircraft in the Air Plan.

For the remainder of this document, the Random agent is referred to as RAND, the Roving Mob agent with Murder All Air Bases agent is referred to as RM-MAAB, and the Roving Mob with Air To Ground Focus is referred to as RM-A2G. These three agents provided a performance baseline to evaluate the performance of the LCS agents.

3.3 LCS Algorithm Development

The experiments consider Multi-Agent Learning Classifier System (MALCS) implementations of the Michigan-style Zeroeth-Level Classifier System (ZCS) and eXtended Classifier System (XCS) algorithms, and a Pittsburgh-style Adaptive Pittsburgh Classifier System (APCS). Each MALCS consisted of two configurations: homogeneous and heterogeneous. All MALCSs share a common semantic structure for rule representation, but differ in learning strategy and action selection.

3.3.1 Rule Representation and Generation

In most LCS systems, rules are mainly represented by fixed-length ternary strings [8]. However, due to the complexity of Stratagem MIST, it would require large memory requirements to form conditions out of the complete state-space and mask human readability. One potential solution is the application of using conjunctions of variable-length condition predicates over continuous values, initially referred to as "messy" representation [52] and later converted to symbolic expressions (s-expressions) [53]. In this scheme, each rule represents a partial component of the state-space, with the ability to be joined together to form a more complete representation of the state-space. For all LCS-based agents, rules are structured using the following structure:

$$Condition\ (\&\ Condition)^* \implies Action\ (Action\ Modifier)^*$$

Each rule contains at least one condition to avoid trivial classifiers, and the number of conditions was determined by a conjunction probability p_c that holds the probability that another rule will be selected for conjunction with the current condition clause after each addition of a condition. For example, let n be the number of conditions in a rule. If $p_c = 0.5$, then $p(n = 1) = 0.5$, $p(n = 2) = 0.25$, $p(n = 3) = 0.125$, until a condition upper bound is reached. Some actions have an action modifier that give further information to the agent about an action, *e.g.* the name of a specific unit to target. The Boolean OR operator is excluded from our scheme due to associated generalization issues, specifically within the XCS algorithm [53]. The Boolean NOT operator is implicitly included in each condition. All implemented LCS algorithms do not have a subsumption component, as subsumption produces smaller rule sets at the cost of adaptability to environmental changes [54] and thus would affect the agent's robustness against different game-playing strategies.

Each LCS agent possesses a separate rule-set each for the ground domain and air domain. Ground conditions are composed of a combination of information relative to the unit, *e.g.* current strength, and information available to all units, *e.g.* total team strength. Thus, a ground unit's inputs are a combination of unique values only specific to the unit to allow individual decision-making as well as shared team inputs that allow for team-based decision making. Air conditions only consist of information available to all units, as air units possess no internal state. A list of available conditions and actions can be found in Appendix B.

3.4 Algorithms Explored

In addition to the baseline agents, the experiments feature three LCS categories and a modern multi-action Genetic Algorithm (GA)-based algorithm known as Online Evolutionary Planning (OEP). Within each LCS category, a homogeneous and a heterogeneous version was explored.

3.4.1 ZCS

For our implementation of the ZCS algorithm in Stratagem MIST, each rule is assigned a strength-based fitness that determines the success of a classifier. In accordance with the initial proposed algorithm of ZCS [29], the genetic algorithm is panmictic, thus taking place in the $[P]$ classifier set. Each classifier (cl) contains its fitness (f) as its only parameter. The credit assignment takes place as follows:

1. Each classifier in the set [M] - [A] (matched classifiers that were not selected) get "taxed" by a fixed fraction τ: $f_{cl} \leftarrow f_{cl}(1-\tau)$.

2. Each classifier in [A] offers a portion of their fitness and places it in a shared "bucket" B: $B \leftarrow B + \beta f_{cl}$, $f_{cl} \leftarrow f_{cl}(1-\beta)$.

3. If an immediate reward ρ exists, then it is reflected in each classifier in [A]: $f_{cl} \leftarrow f_{cl} + \frac{\rho\beta}{|A|}$.

4. All classifiers in the previous action set $[A]_{-1}$ are updated: $f_{cl} \leftarrow f_{cl} + \frac{\gamma B}{[A]_{-1}}$.

3.4.2 XCS

Our application of the XCS algorithm in Stratagem is modeled off Butz and Wilson's refined XCS algorithm [54]. In this implementation, the genetic algorithm is performed using only the classifiers in the action set. Notable deviations from the core algorithm include the use of a power law function to calculate the accuracy κ, and the addition of a parameter that controls the minimum number of classifiers in a match set before a covering mechanism occurs. For each possible action A, XCS contains a prediction array that predicts the reward R. Each classifier (cl) contains four distinct parameters: prediction (p), prediction error (ϵ), fitness (f), and experience (exp). The parameters are updated as follows: [8]

1. The ϵ in each classifier is updated: $\epsilon_{cl} \leftarrow \epsilon_{cl} + \beta(|R - p_{cl}| - \epsilon_{cl})$.
2. Each prediction is updated: $p_{cl} \leftarrow p_{cl} + \beta(R - p_{cl})$.
3. The accuracy is computed: $\kappa = \epsilon_{cl}^{-exp}$ if $\epsilon_{cl} > \epsilon_0$, otherwise 1.
4. A relative accuracy is computed: $\kappa'_{cl} = \frac{\kappa_{cl}}{\Sigma\kappa[A]_{t-1}}$.
5. Each fitness is updated: $f_{cl} \leftarrow f_{cl} + \beta(\kappa'_{cl} - f_{cl})$.
6. The experience exp is incremented for all classifiers: $exp \leftarrow exp + 1$.

3.4.3 APCS

The experiments evaluate a multi-agent version of APCS, described in [37]. APCS is primarily based off of Smith's initial Pittsburgh-style algorithm [28]. It consists of:

1. A population of fixed-size rule-sets are randomly initialized and its fitness is set to zero.
2. Each individual trains for k iterations using a fixed action selection policy. Upon receiving a reward, it is added to the rule-set's fitness.
3. A GA component creates the next generation offline, and fitness is reset to 0.

Enée [37] notes four different action selection policies for APCS. In the *First/F&F* policy, the rule-set is ordered in a decision list such that rules at the top of the list are prioritized over lower-ranked rules. The remaining action selection policies utilize unranked rule-sets. In the *Specific* selection policy, the most specific rule, *i.e.*, the rule with the most condition constraints, is selected to create tuned decision-making to small changes in the environment. In the *General* selection policy, the most general rule, *i.e.* the rule with the least condition constraints, is selected to create generalized decision making. Finally, in the *Random* selection policy, rules are randomly selected such that any matched rule in the rule-set has equal chance in being picked.

There are advantages to each policy. The First/F&F action selection method is easily human-readable as it corresponds to a decision list. The General policy is especially of interest in Stratagem MIST, as domains can vary vastly across scenarios. This method could theoretically create adaptable classifiers that scale across unknown scenarios. In contrast, the Specific policy produces a tuned rule-set to a specific scenario if known beforehand. Finally the Random policy attempts to enforce that all rules in the rule-set lead to a desirable state, as any matched classifier could be selected and thus harshly penalizes negative mutations. To the best of the author's knowledge, no effort has been made to compare action selection policies or distinguish which policy corresponds to a specific domain. Section 4.1.1 provides a comparison of the policies for Stratagem MIST.

For the GA component, our implementation of APCS uses uniform crossover between two rule-sets, with each rule representing an allele. A mutation rate of 0.1 is considered at each allele to promote diversity. The APCS preserves its fittest individual via elitist strategy and Tournament-Selection (k=5) to form the next generation in its GA. Figure 6 demonstrates this process.

It is important to note that unlike the Michigan-style algorithms, all learning occurs offline, and thus APCS requires significantly longer training times. However, unlike ZCS or XCS, the resulting decision list is a readable optimized instruction list

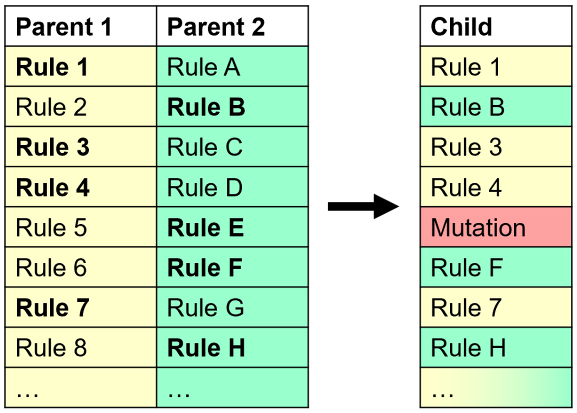


Figure 6: Crossover for APCS.

that requires no or minimal additional computation to perform action selection.

3.4.4 Heterogeneous LCS

As Stratagem MIST is a novel multi-agent domain, homogeneous and heterogeneous strategies have yet to be explored. To verify the success of LCS agents under both options, all examined MALCS agents (ZCS, XCS, and APCS) were separated into homogeneous/heterogeneous pairs. For the remainder of this document, the heterogeneous versions are referred to by an Ht prefix, while the homogeneous versions are referred to by their base name, as the original multi-agent versions [47, 48] were homogeneous. In the homogeneous version, all units in the ground component of the game share a centralized rule-set and thus perform actions based off the same rules. To prevent two units from always performing the same action in the same position and encouraging diversity in actions, some rules are only accessible to certain units or unit types. For example, a condition set may be joined with the condition *type = "Light Infantry"* or *unit = "Fire Team Alpha"*. In the air component of the game, rules are only based off non-relative information, and each action directly affects the complete air plan. The immediate benefit of this strategy is lower training time, evaluating one set of rules at a time as opposed to sets of sets of rules.

In the heterogeneous MALCS agents, each ground unit is assigned their own rule-set. This rule-set is tied to a unit's unique identifier, such that rule-sets cannot be carried over between scenarios, as units possess different names and types in other scenarios. In the heterogeneous Michigan-style MALCSs, the GA still occurs within each individual's rule-set, and does not select parents between units. In Heterogeneous Adaptive Pittsburgh Classifier System (HtAPCS), crossover can occur between individuals, but occurs only between rule-sets with the same identifier. Therefore, rules and rule-sets only crossover with matching identifiers in other individuals (i.e. Unit A cannot crossover rules with Unit B in another population). An example iteration can be found in Figure 7.

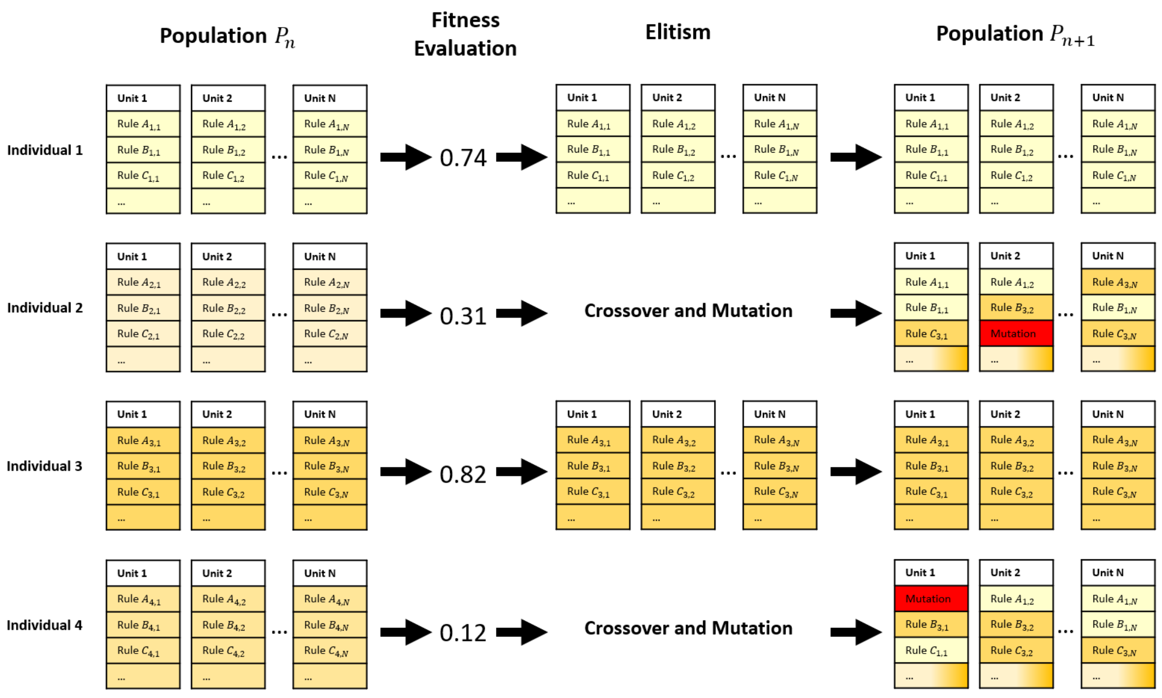


Figure 7: A sample HtAPCS Iteration with four individuals and elitism($n = 2$). The fittest individuals are preserved, with the least fit individuals formed from a crossover of the elites with mutation. Unit Identification is preserved in this crossover. In other words, a child's Unit 1 must be formed from either Unit 1s of its parents (barring mutation).

3.4.5 Online Evolutionary Planning

The aforementioned baseline and LCS agents also competed against an agent using OEP, a modern non-symbolic GA-based learning algorithm. Designed by Justesen [50], OEP is a multi-action game-playing algorithm that uses a genetic algorithm to determine the best action. The initial population consists of vectors of randomly-selected actions. Each vector is passed through an evaluation process and assigned a fitness score. A GA then crossovers and mutates the fittest individuals to form the next generation and repeat the cycle. The OEP algorithm's notable attribute is that it only focuses on developing an action sequence for the current turn instead of developing a "rolling horizon" action list for subsequent turns. The psuedocode for the algorithm can be found in Algorithm 2. Our implementation matches the original algorithm with the exception that the evaluation function uses a depth-limited roll-out instead of a static state scoring function to avoid greedy states and evaluate short-term consequences of a certain move.

Algorithm 2 The Online Evolutionary Planning algorithm [50].

```
    numPop = # of Individuals in Population
    numIt = # of Iterations
    pMut = probability of Mutation
    state = current game state
 1: function OEP(numPop, numIt, pMut, state)
 2:     ind[] pop = population of size numPop seeded with random actions;
 3:     count = 0;
 4:     while count < numIt; do
 5:         for Individual ind in pop do
 6:             clone = CLONE(state);
 7:             clone.update(ind.actions);
 8:             if ind.visits == 0; then
 9:                 ind.value = EVAL(clone);
10:             end if
11:             ind.visits++;
12:         end for
13:         pop.GA(pMut);
14:         count++;
15:     end while
16: end function
```

3.5 Stratagem MIST Environment

All agents competed on Stratagem MIST Release 4's default scenarios:

- Hokkaido simulates a conflict taking place on the northernmost of Japan's main islands. The objective is to inflict the most damage on the opponent and prevent units from taking damage.

- Hex Battle simulates a conflict on a 10 x 9 grid where most of the nodes have a degree of six. The objective is to receive the most points for capturing and holding six nodes in the grid

- Three Main Lanes simulates a conflict where two opposing teams must destroy each other's capital, and may travel through three lanes.

- Air Assault On Crete simulates a conflict on the Greek island of Crete. Like the

Hokkaido scenario, the objective is to inflict the most damage on the opponent and prevent units from taking damage.

All starting configurations in the default scenarios are shown in Figure 8. Scenario-specific information regarding units and map layouts can be found in the appendix (Table B.5).

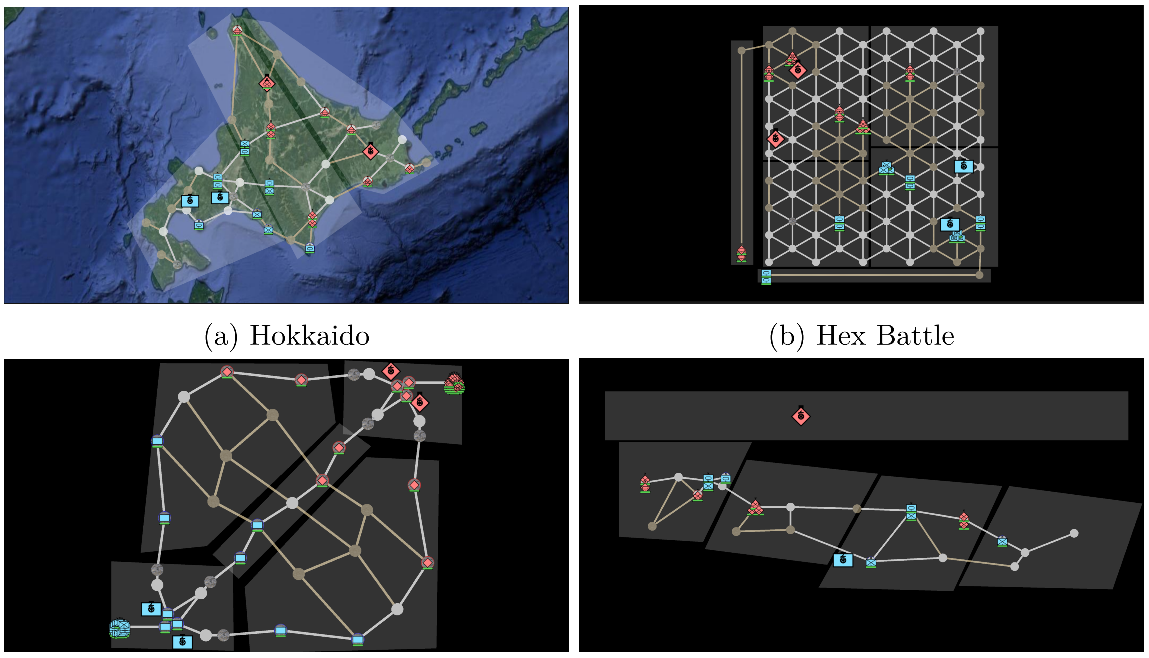

(a) Hokkaido (b) Hex Battle (c) Three Main Lanes (d) Air Assault On Crete

Figure 8: Stratagem MIST's four default scenarios

3.6 Reward Evaluation and Performance Metrics

Stratagem MIST has a built in scoring function that evaluates the current metrics and returns a scalar value. In all scenarios except Three Main Lanes due to its use of objectives in addition to metrics, this scoring function is applied at the end of the game to determine the winner. In Three Main Lanes, this evaluation is applied only if neither teams' objective is not completed. All agents use a version of this scoring function as a measure of fitness opposed to win rate, with a win/loss bonus to help distinguish between actions that lead to a marginal victory and a clear victory. Thus, the reward for all agents is calculated as shown in Algorithm 3. In Chapter IV, the win rate and the pure metric evaluation are used as a performance metric for all agents.

Algorithm 3 Reward Function for Agents

```
$O_f$ = Set of Friendly Objectives
$O_e$ = Set of Enemy Objectives
$M_f$ = Set of Friendly Metrics
$M_e$ = Set of Enemy Metrics
$W$ = Win/Loss Bonus
function GenerateReward($O_f, O_e, M_f, M_e, B$)
    if $O_f$.AllCompleted() then
        return 1 + $B$
    end if
    if $O_e$.AllCompleted() then
        return -1 - $B$
    end if
    score ← METRICEVALUATION($M_f, M_e$)
    if score > 0 then
        score ← score + $B$
    else if score < 0 then
        score ← score - $B$
    end if
    return score
end function

function MetricEvaluation($M_f$,$M_e$)
    myScore ← 0
    enemyScore ← 0
    for m in $M_f$ do
        myScore ← myScore + m.value * m.multiplier
    end for
    for m in $M_e$ do
        enemyScore ← enemyScore + m.value * m.multiplier
    end for
    return myScore - enemyScore
end function
```

IV. Results and Analysis

This chapter presents the experiments designed to answer the research questions and the associated results. Section 4.1 details the design of the conducted experiments. Section 4.2 displays the results of the main Round-Robin tournament and analyzes a supplemental general case. Section 4.3 examines each agent's ruleset composition and how it relates to behavior. Section 4.4 looks into how each Learning Classifier System (LCS) prompts interpretablity. Finally, Section 4.5 discusses limitations and solutions to the explainability of the agent's rule-sets.

4.1 Design Decisions

This paper identifies two potential gaps in research involving LCSs that require experimentation prior to Round-Robin comparison. The first is a direct comparison of action selection strategies for Pittsburgh-style LCSs. The second is an analysis of setting up rule-sets for Multi-Agent Learning Classifier Systems (MALCSs). Because these are preliminary experiments, only agents that do not require any training must be used for testing. For the remainder of the document, we define groups of agents as the following:

- Baseline agents: RAND, RM-MAAB, and RM-A2G.
- Non-LCS agents: The baseline agents plus the Online Evolutionary Planning (OEP) agent.
- Homogeneous LCS agents: Zeroeth-Level Classifier System (ZCS), eXtended Classifier System (XCS), and Adaptive Pittsburgh Classifier System (APCS).
- Heterogeneous LCS agents: Heterogeneous Zeroeth-Level Classifier System (HtZCS),

Heterogeneous eXtended Classifier System (HtXCS), and Heterogeneous Adaptive Pittsburgh Classifier System (HtAPCS).

- LCS agents: All agents that utilize a LCS.

4.1.1 APCS Action Selection Comparison

The APCS algorithm is the least occurring algorithm in literature, and therefore required preliminary experimentation prior to testing. This experiment compared the action selection policies described in Section 3.4.3, in addition to a fifth selection policy: Mode. This action selection policy takes the most advocated action in the match set. If there is a tie, or if the action modifiers do not match, placement in the rule-set is the tie-breaker. The experiment did not consider the base maps as winning strategies could be found by the APCS agents against the baseline agents after only a few generations. Therefore, the agents trained on custom "Hard" version of Hokkaido, which involved the removal of some friendly ground units and the addition of enemy ground units. Each individual played as the red player to avoid winning by tie and ran 100 games against an even rotation of the four non-LCS agents. The first generation's rule-set was randomly initialized with 100 ground rules and 100 air rules. The Genetic Algorithm (GA) formed each subsequent rule-set through elitist preservation (n=1) and tournament selection (k=5). The average scores of each individual in the population in each generation across 10 trials can be viewed in Figure 9.

Across all scenarios, the Fast and Frugal method, Specific method, and General method prevailed over the Mode method and the Random method. The means of the top three methods all fall within each others' 95% confidence intervals in the latter generations, and thus any one of them could be the most effective method in this domain.

For the remainder of experiments, the Fast and Frugal method provides the action selection policy used by the APCS and HtAPCS agents, as its decision-list nature is highly interpretable [19] and does not require iterating through all rules in the rule-set as the other two methods would.

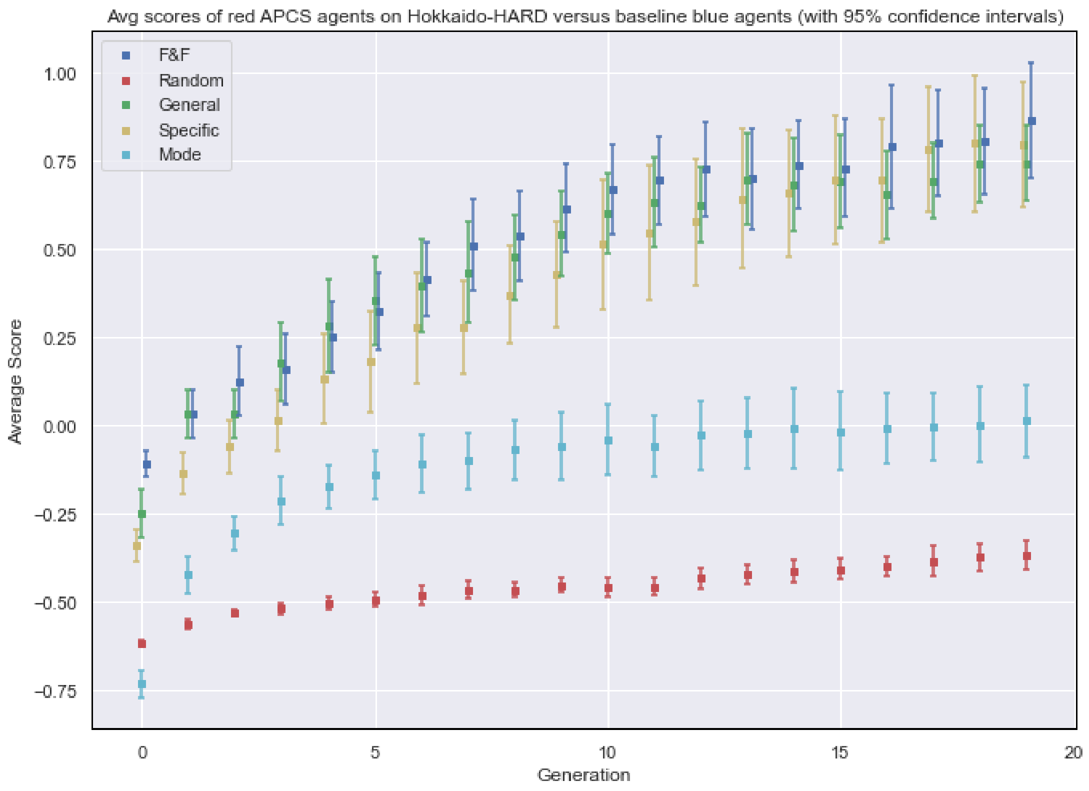


Figure 9: The average score over 10 trials each of red APCS agents versus baseline blue agents using different action selection policies. Error bars are 95% confidence intervals of the means.

4.1.2 Standardizing Starting Rule-sets for Heterogeneous Agents

Early implementations of non-baseline agents would often fail if units were initialized to randomly move independently of each other. While the agents would eventually find the winning strategy, it would take significantly longer to converge. This effect can be attributed to a strength-in-numbers effect in Stratagem MIST, where clusters of units are less vulnerable to enemy attacks. Therefore, a method to catalyze the convergence process is to seed the initial population with coordinated movements. For the OEP agent, a individual where each unit initially moves to the same node is guaranteed to be in the initial population. As crossover and mutation occurs, deviations from this strategy allow for units to diverge from the main group to find optimal strategies. For homogeneous LCS agents, this problem does not exist due to the pre-built *Converge* action, which encourages unit grouping if the condition is general enough. However, for the heterogeneous LCS, agents have a small probability of initially clumping in groups. A solution to this problem is a two-phased approach where all ground units initially follow a shared rule-set, and then allow for individual crossover and mutation to become heterogeneous. Once the agent trains for k iterations and converges on a homogeneous strategy, units are free to find unit-specific rules that improve team performance. For our implementation, k was set to 1.

The results of a preliminary experiment to test this strategy can be found in Figure 10, which plots the results of red heterogeneous agents against blue non-LCS agents on the Air Assault on Crete scenario. In the experiment, agents are assigned the category NS (Non-standardized, agents were initially assigned different rule-sets) or S (Standardized, agents were initially assigned a single rule-set). The Pittsburgh-style agents used the Fast and Frugal action selection method. All initially standardized agents converged faster than their non-standardized counterparts, although with different margins, as noted in Table 3. As two S-LCS agents are found to outperform their NS-LCS counterpart, the remainder of experiments only utilized standardized agents.

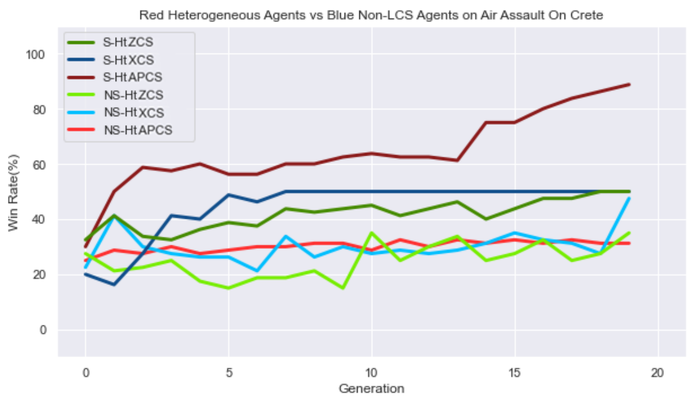


Figure 10: The standardization strategy is denoted by the prefix NS (Non-standardized, agents were initially assigned different rule-sets) or S (Standardized, agents were initially assigned a single rule-set).

	Mean	σ	**Max**	**p-value**	**t-value**	**Statistically Significant**
S-HtZCS vs. NS-HtZCS	0.221/0.161	0.155/0.082	0.500/0.350	.274	-1.12	No
S-HtXCS vs. NS-HtXCS	0.465/0.194	0.047/0.107	0.500/0.475	<.00001	-7.680	Yes
S-HtAPCS vs. NS-HtAPCS	0.699/0.274	0.241/0.018	0.898/0.313	<.00001	-5.839	Yes

Table 3: Results of a t-test comparing standardized vs. non-standardized implementations on Air Assault On Crete.

4.1.3 Performance Comparison

To assess the efficiency of all agents, a Round-Robin tournament was performed. The tournament compared games between ten agents: the three baseline agents described in Section 3.2, the three homogeneous LCS agents, the three heterogeneous LCS agents, and an agent using the OEP method described in Section 3.4.5. An agent played every agent (including themselves) across the four default maps as both the red and blue teams. In addition, to test the ability of the LCS agents to perform generally, LCS agents were also tested in a General case, where they are trained on all scenarios and thus are barred from using scenario-specific conditions and actions.

4.1.4 Training and Testing Information

There exists a challenge training the Michigan-style agents against the Pittsburgh-style agents due to the differences in learning strategies (online vs. offline). Under default training conditions, the Michigan-style agent would be inherently advantaged because it could train against a population of individuals and thus train against diverse play-styles. An alternative would be to train the Michigan-style agents as a population as opposed to a single individual. Therefore, the experiments train a population of Michigan-style MALCSs.

Section 2.2.1 outlined four methods to co-evolve agents proposed in [11]. A variation of the first approach was chosen, which consists of individuals competing against randomly selected individuals in opposing populations. Thus, the training scheme consists of twelve (six red/six blue) homogeneous populations consisting of agents using each of the algorithms. As each individual learns or is evaluated, it is paired with a randomly-selected individual in the opposing population. The four non-LCS agents (RAND, RM-MAAB, RM-A2G, and OEP) are considered in evaluation but do not contain their own populations due to no variation across the agents. Populations consisted of 20 individuals. Upon forming a new generation, the 10 fittest individuals were preserved and the new generation was formed from the elites via uniform crossover with a mutation rate of 0.02. LCSs possess many different parameter settings. Due to the novelty of Stratagem MIST's domain, no predefined LCS parameters exist to immediately adapt into the tournament. Therefore, a GA was utilized to tune the necessary parameters for the agents. A description of these parameters and their values are shown in Table 4.

Upon the completion of training, each individual in the population is subject to an evaluation of a preliminary 200 games against a random sample of the opposing population. The fittest individual from each population is chosen to compete in the

Parameter	**Description**	**ZCS**	**XCS**	**APCS**
N_G	# of ground rules in population	1000	1000	100
N_A	# of air rules in population	1000	1000	100
α	Controls the decline in accuracy if classifier is inaccurate	N/A	.025	N/A
β	Learning rate	.025	.025	N/A
γ	Discount factor	.8	.8	N/A
ϵ	Probability of selecting a random action (exploration)	.05	.05	N/A
θ_{del}	Minimum threshold required for deletion vote	20	20	N/A
μ	Probability of mutation in GA	.3	.3	.3
ν	Fitness exponent	N/A	5	N/A
τ	Strength deduction for non-selected classifiers	.001	N/A	N/A
χ	Crossover rate in GA	.25	.25	.25
e_0	Minimum error for classifiers to be considered of equal accuracy	N/A	.2	N/A
p_c	Conjunction probability	.8	.8	.5
p_0	Minimum number of conditions required in condition clause	1	1	3
p_{GA}	Probability of GA in each step	.05	.01	N/A

Table 4: Values for LCS parameters for all implementations. Homogeneous and Heterogeneous agents used the same parameters.

Round-Robin tournament. Each agent then plays 1000 games against each agent (including itself) on each scenario. All training/testing was shared between an Intel Xeon Core CPU E5-2687W with 3.10GHz clock speed/64GB RAM and an Intel Core i7-10700 with 2.90 GHz clock speed/16GB RAM.

4.2 Main Tournament Results

The win rates and final scores for each agent match-up on all scenarios can be found in Appendix A. Aggregated win rates across all scenarios are shown in Table 5, along with a relative rank associated with the combined win rates of all scenarios. While the LCS agents utilized a LCS in both the ground and air domains, the ground domain was mainly analyzed as it is the primary domain. The four default scenarios are not balanced, even though they start off with the same assets. The blue team has a default advantage due to ties in the scoring system result as a blue win. The blue agent may use this feature to their advantage, achieving a win by picking a strategy that ultimately leads to a scoring tie. As an example, this is evident in blue HtAPCS versus red XCS matchup on Hokkaido (see Table A.1 and Table A.2). While every

Red/Blue	Hokkaido		Hex Battle		Three Main Lanes		Air Assault On Crete		Combined		Relative Rank	
RAND	4.6	6.3	8.2	5.6	38.2	43.3	0.9	52.1	12.9	26.8	10	10
RM-MAAB	13.3	44.4	49.1	20.2	0	20	0.1	59.7	15.6	36.1	8	9
RM-A2G	13.2	44.3	49.2	26.4	0	20	0.1	59.6	15.6	37.5	8	8
OEP	39.7	50.4	32.5	25.3	52.6	53.7	14.4	83.2	34.4	53.1	7	6
ZCS	79.2	70	87.5	62.1	77.3	64.6	37.4	82.3	69.4	69.7	3	5
XCS	46.1	80.6	52.4	39.1	21.1	73.5	22.2	95.1	34.9	72.1	6	4
APCS	82.2	100	69.4	79.1	73.5	60	44.9	100	71.4	84.8	**1**	**1**
HtZCS	43.3	73.2	73.4	74.4	71.4	75.5	27.6	86.8	53.2	77.5	4	2
HtXCS	21	29.4	65.6	21.7	74.8	54.1	3.7	99.5	41.2	51.2	5	7
HtAPCS	89.7	69.2	77.9	81.4	67.4	59.2	50.4	100	70.1	77.4	2	3
Average	43.2	56.8	56.5	43.5	47.6	52.4	20.2	79.8				

Table 5: The win rates for all scenarios against every other agent in the tournament. The win rates are presented as (Red/Blue), where the left value is the overall win rate of the red agent and the right value is the overall win rate of the blue agent. In addition, a combined score of the four scenarios is presented along with a relative rank based on the combined score.

game in this configuration technically resulted in a scoring tie, the blue agent won every game. The weight of this advantage is dependent by scenario. For example, under a red RAND vs. blue RAND configuration, 7.41% of games on Hokkaido resulted in a tie, while 0.47% of Hex Battle games resulted in a tie.

The ZCS agent performed average among the tested agents, relatively ranking 3rd as a red player and 5th as the blue player. The HtZCS agent achieved similar performance, relatively 4th as the red player and 2nd as the blue player. Across all scenarios, it was able to achieve a 100% win rate against the Roving Mob agents and 100% or near 100% against the RAND agent. The XCS and HtXCS agents under-performed their LCS counterparts, respectively ranking 6th and 5th on the red team and 4th and 7th on the blue team. On the Air Assault on Crete Scenario, the HtXCS agent was unable to outperform any blue agent. The APCS agent performed well, relatively ranking 1st on both teams. The HtAPCS agent achieved similar performance, relatively ranking 3rd on the blue team and 2nd on the red team. These implementations were also the only agents to achieve 100% win rates against all other agents on a scenario (Blue APCS on Hokkaido and Air Assault On Crete, and Blue

HtAPCS on Air Assault On Crete). The results indicate that both agents are able to succeed under different environmental circumstances, showing their effectiveness as complex game-playing agents.

4.2.1 General Case

One metric of an agent's success in Stratagem MIST is their ability to perform with no prior knowledge of the environment. We redefine the objective to create a general agent that can apply its rule-sets to any scenario, as Stratagem Release 4 has limited capability to create new maps outside of the default four. To test this ability, a supplementary experiment where agents cannot use scenario-specific knowledge was performed in addition to the main tournament. Under these constraints, agents can only use scenario-general rules and are barred from using any condition or action that use a specific aspect of the environment, e.g., Move to Node "X" on scenario "Y". Thus, classifiers can only contain condition and actions labeled as "General" in Tables B.1 and B.2. This experiment only considers the homogeneous agents due to heterogeneous agents assigning rule-sets by unique unit identifiers that do not transfer across scenarios. The non-LCS agents are unaffected, as they already possess no knowledge of the environment or optimized-strategy tailored to an environment prior to the start of the game.

To train the agents, each LCS agent trained against the baseline agents and populations of the LCS agents on each of the four scenarios on both sides. Each agent played all agents (including itself) for 100 games on each scenario. The aggregated results of each agent, along with a relative rank among agents, are shown in Table 6.

The inclusion of this case measures the ability of the agents to perform when the environment is not explicitly given. Overall, the agents did not perform nearly as well without the scenario-specific classifiers but were still able to outperform the

	RAND	**RM-MAAB**	**RM-A2G**	**OEP**	**ZCS**	**XCS**	**APCS**
Win Rate	42.1	38.3	37.4	73.4	59.4	45.4	54.0
Relative Rank	5	6	7	1	2	4	3

Table 6: The aggregated win rates of agents across all scenarios. The LCS agents were limited to scenario-general rules. A relative rank is calculated among the agents.

baseline agents. They were not able to outperform the OEP agent, which achieved a 73.4% overall win rate. This shortcoming is likely due to the play-forward ability of OEP which can help on an unseen scenario. While the perceived adaptability of the agents in the general case may be due to the limited number of scenarios available to train on, as there exist a countably infinite set of environment configurations, a larger sample size is required to further analyze the ability of agents to generalize.

4.3 Rule Composition and Behavior Analysis

To the best of the author's knowledge, LCSs within the wargame subclass of games are absent in literature. Therefore, to determine how the algorithms approach wargaming, a deep dive into the internal makeup of the rules is warranted. This section observes the internal makeup of the rule-sets with respect to condition/action distribution and selection frequencies.

4.3.1 ZCS

A look into the action selection behaviors of a few of its games reveal much about the nature of the ZCS agent. Figure 11 compares the firing of classifiers for red ZCS and HtZCS agents across three games against a blue RAND agent on Hex Battle. A classifier is considered "fired" if it is in [A] at the end of a step. The ZCS agent only shared eleven of its 500 classifiers across all three games, and there exist between 9-30 classifiers only used in one or two games. However, the "base" classifiers were fired the most often, with the eleven classifiers accounting for 84.0% of the agent's

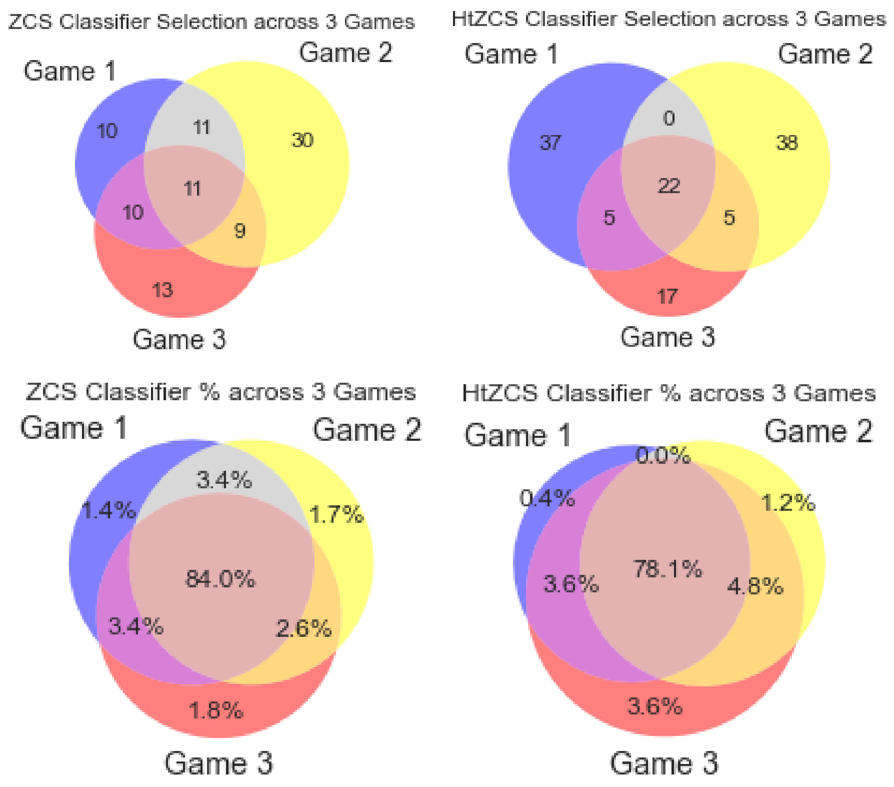


Figure 11: A comparison of the sets of classifiers chosen by three different trials of a red ZCS and HtZCS versus a blue RAND agent on Hex Battle. The top diagrams track the subsets of classifiers across the trials, while the bottom graphs track their frequency in action selection.

action set throughout the game. This behavior suggests that in Stratagem MIST, high performance can be achieved with a small set of actions. A supplementary experiment tracking the firing of classifiers of a blue ZCS agent against a red RAND agent on Hex Battle across 100 games discovered that the agent utilized 100% of its classifiers at least once (all classifiers at least advanced to [M]), fired 30.2% of its available classifiers, but fired its top 3 classifiers 80.6% of the time. This is displayed in Figure 12, which shows the cumulative distribution of the fired classifiers across all games. It is important to note that the top three classifiers advocate for the same action, G11-Urban: move to the nearest urban terrain, or defend if already there. This strategy makes sense in the context of the game, as urban terrain provides large

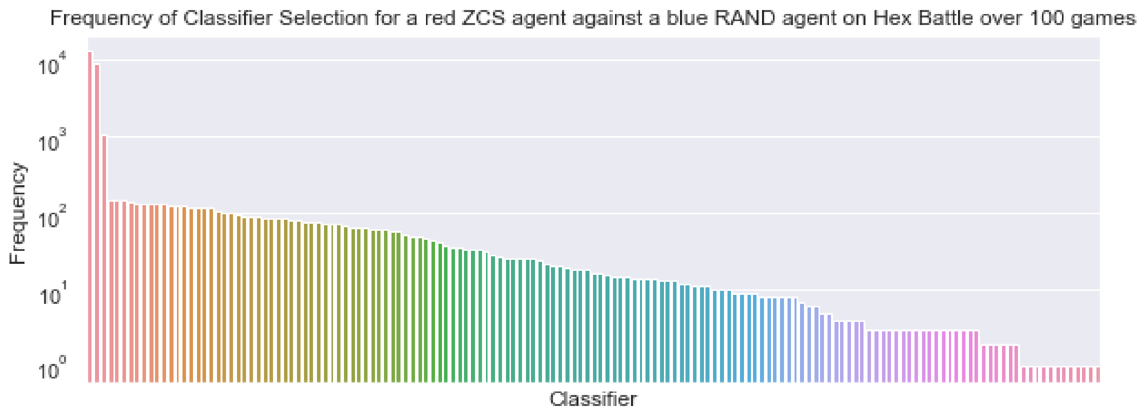


Figure 12: The distribution of classifiers in the finalized action set across 100 games of a red ZCS agent against a blue RAND agent on Hex Battle.

defense multipliers for certain units. An outside observer would note ZCS's initial strategy is to move its forces to the nearest urban terrain (which all happen to be point-generating nodes), defend to generate points and protect against aggressive agents, and then seek further point-generating nodes in the late-game. However, the turn that the agent switched to its node-seeking phase varied between games and showed dependence of the behaviour of the opposing agent. This phenomenon highlights how ZCS can react to environmental changes.

4.3.2 XCS

XCS achieved a lower win rate than ZCS. A glimpse into its classifier selection can be offered as an explanation. Figure 13 tracks the classifier selection across three games on Hex Battle against a RAND agent. XCS possessed no deviation in classifiers across its games, while HtXCS possessed some deviation but less of an extent than ZCS. Each turn, the XCS agent ordered units to converge to the nearest urban terrain, but did not initiate any other order regardless of state change. This proved to be a less effective strategy, as the red ZCS agent outperformed its XCS counterpart against five agents and only under-performed against one agent. These

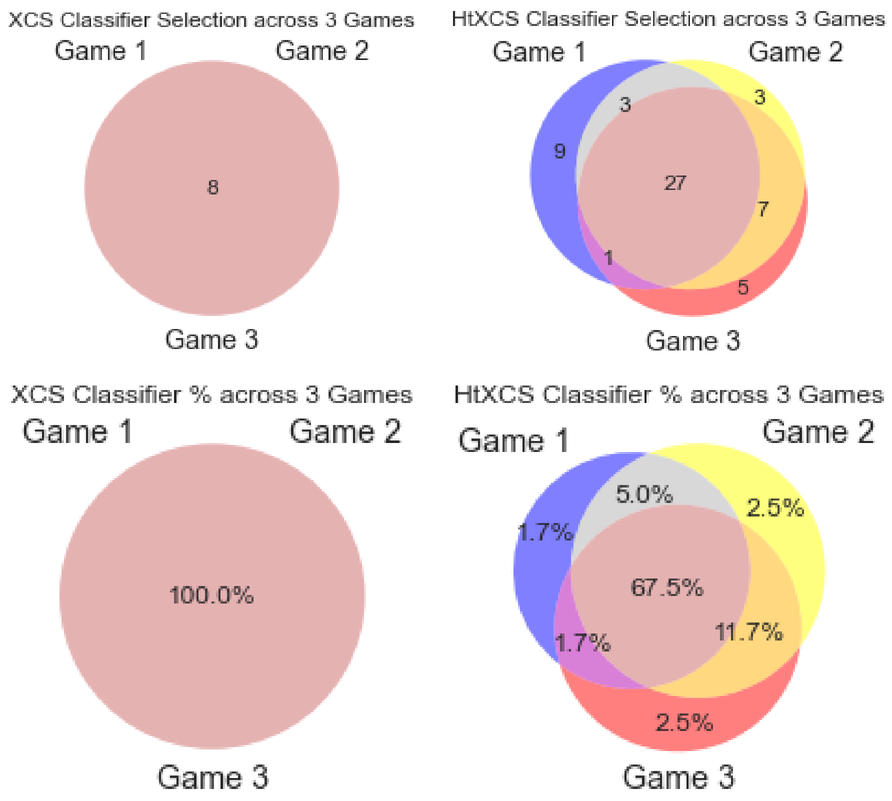


Figure 13: A comparison of the sets of classifiers chosen by three different trials of a red XCS and HtZCS versus a blue RAND agent on Hex Battle.

XCS agent likely performed worse than ZCS due to high error values on states closer to the beginning of the game, upon which the prediction is heavily dependent of the agent currently being played. If the XCS agent develops a winning strategy against a single agent but loses to a secondary agent, its fittest classifiers are at risk due to the XCS's credit assignment scheme. A classifier that previously had a high fitness is punished twice with a lower prediction and a higher error. ZCS does not suffer from this problem as while a negative reward would lower the fitness, but would gradually grow over time assuming the number of positive rewards is greater than the number of negative rewards. While subsequent experiments showed that XCS can develop a winning strategy if trained only on a single baseline agent, it struggles when training on multiple agents. This is likely due to high and volatile prediction errors, as observed in Figure 14. A potential solution is an adaption to an offline

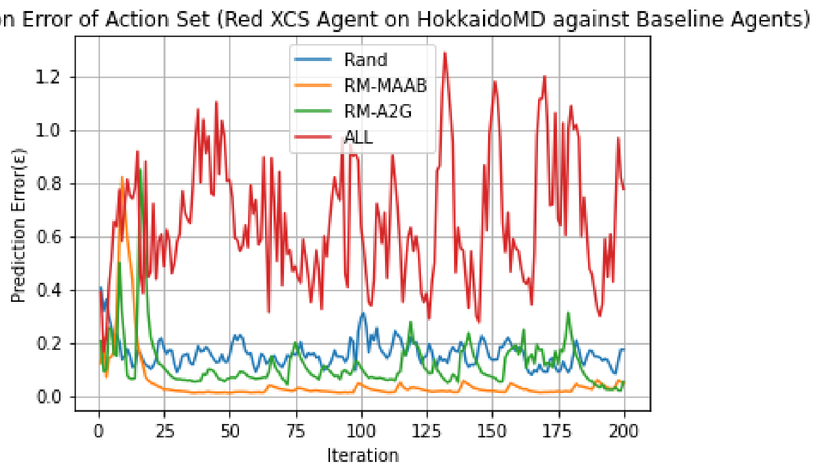


Figure 14: The average prediction error of all classifiers in the action set of a red XCS agent on HokkaidoMD over 200 games. The agent played against each of the baseline agents, in addition to a rotation of all three agents (denoted as ALL). When learning against all agents, the average prediction error in the action set was larger and more volatile. A large prediction error negatively impacts XCS's ability to find optimal classifiers.

learning system, similar to APCS, where the reward is delayed and thus averaged over the course of multiple games.

4.3.3 APCS

To visualize the rule selection process for these agents, we tracked games featuring APCS and HtAPCS agents against a RAND agent under each configuration (each side on each scenario). Figures 16 to 19 show the results of the red APCS and HtAPCS agents against a blue RAND agent on each scenario. Figure 15 displays the cumulative action selection distribution of the the sixteen tracked games. Over the course of the games, the agents showed preference or aversion to certain actions, but no specific action dominated the distribution. This trend is of interest because APCS outperformed the other agents while taking very different actions, showing its adaptability and ability to use all available actions in its decision-making process.

Scenario Name	**Red Agent**	**Blue Agent**	**# Rules Utilized**	**Avg Rule Position**	**Standard Deviation**	**Avg Distinct Rules Per Turn**
Hokkaido	APCS	RAND	9	25.6	9.3	2.4
Hokkaido	HtAPCS	RAND	10	8.5	6.7	2.8
Hokkaido	RAND	APCS	3	6.0	4.6	1.2
Hokkaido	RAND	HtAPCS	7	28.0	15.6	1.4
Hex Battle	APCS	RAND	6	13.3	13.0	2.0
Hex Battle	HtAPCS	RAND	15	26.3	29.1	4.6
Hex Battle	RAND	APCS	5	12.4	8.1	2.5
Hex Battle	RAND	HtAPCS	12	28.9	22.2	2.3
Three Main Lanes	APCS	RAND	7	18.0	10.6	2.1
Three Main Lanes	HtAPCS	RAND	16	27.9	21.0	4.1
Three Main Lanes	RAND	APCS	9	28.2	30.4	1.5
Three Main Lanes	RAND	HtAPCS	22	25.5	26.6	5.7
Air Assault On Crete	APCS	RAND	2	12.0	11.3	1.0
Air Assault On Crete	HtAPCS	RAND	3	27.7	22.0	1.1
Air Assault On Crete	RAND	APCS	4	35.5	23.4	1.2
Air Assault On Crete	RAND	HtAPCS	11	47.1	32.6	3.3

Table 7: Information regarding the resulting APCS and HtAPCS rule-sets across all scenarios. In this experiment, all agents only played one game per configuration.

Table 7 displays information regarding the utilized rules across all tables. In a single game, no Pittsburgh-style agent used more than 22% of the rules in their rule-sets. There also appears to be no distinct advantage or disadvantage to having general rules near the top of the list as opposed to spreading classifiers among classifiers that never fire. On average, the Pittsburgh-style agents for Three Main Lanes used 13.5 rules while the agents for Air Assault on Crete used 5.0 rules. Furthermore, the Air Assault on Crete scenario on average contained less variation in movements compared to the other scenarios. This phenomenon suggests that more complex environments foster more complex behaviors.

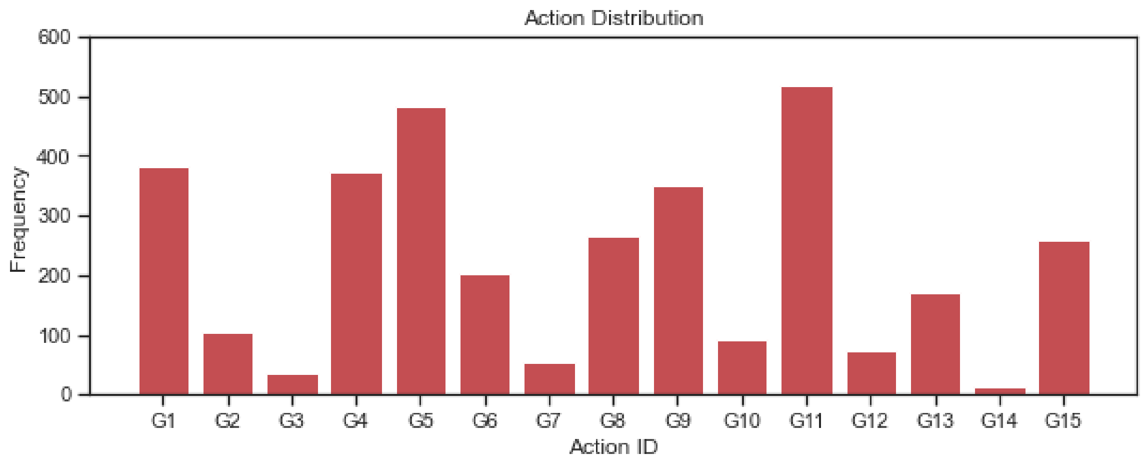


Figure 15: The cumulative action distribution of all fired rules across all heterogeneous and homogeneous APCS configurations across all scenarios.

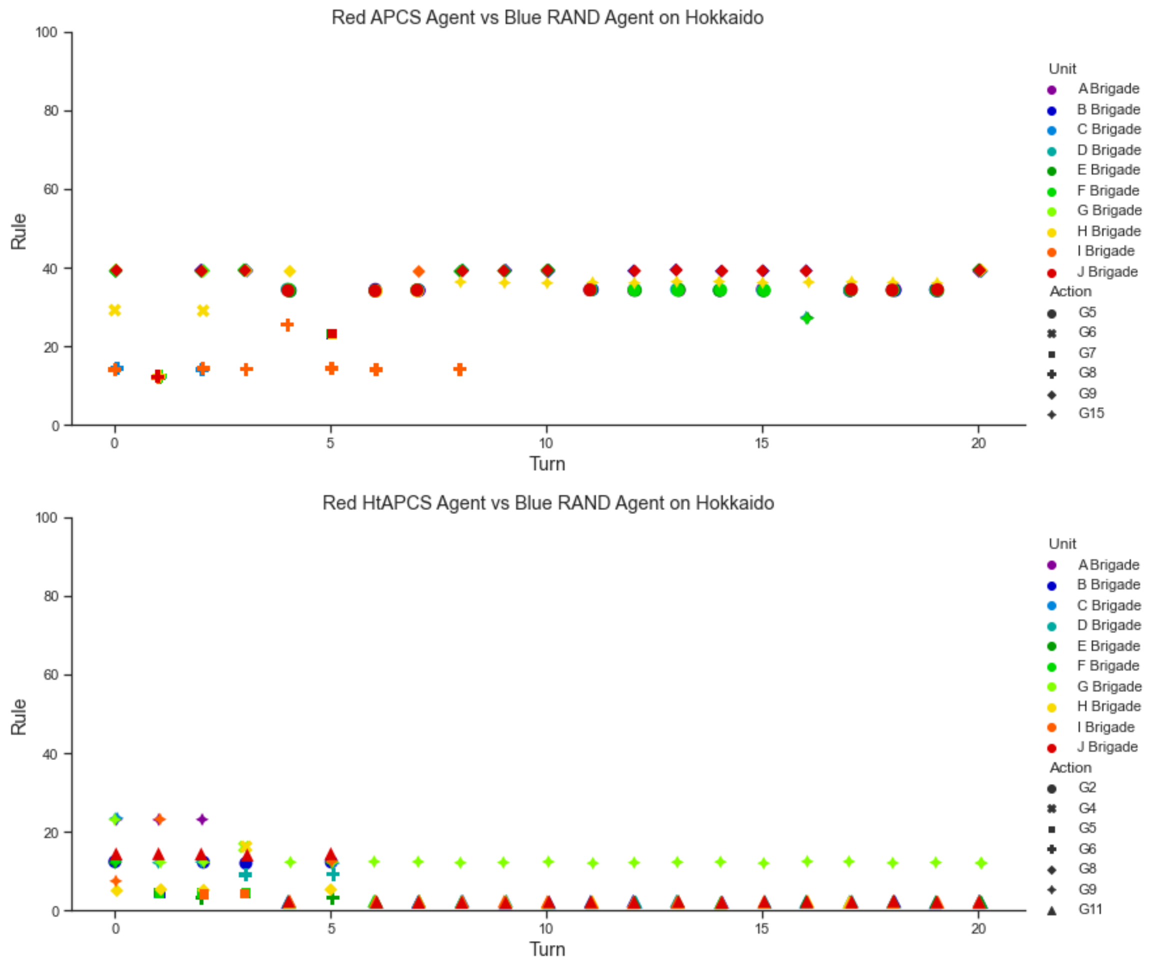
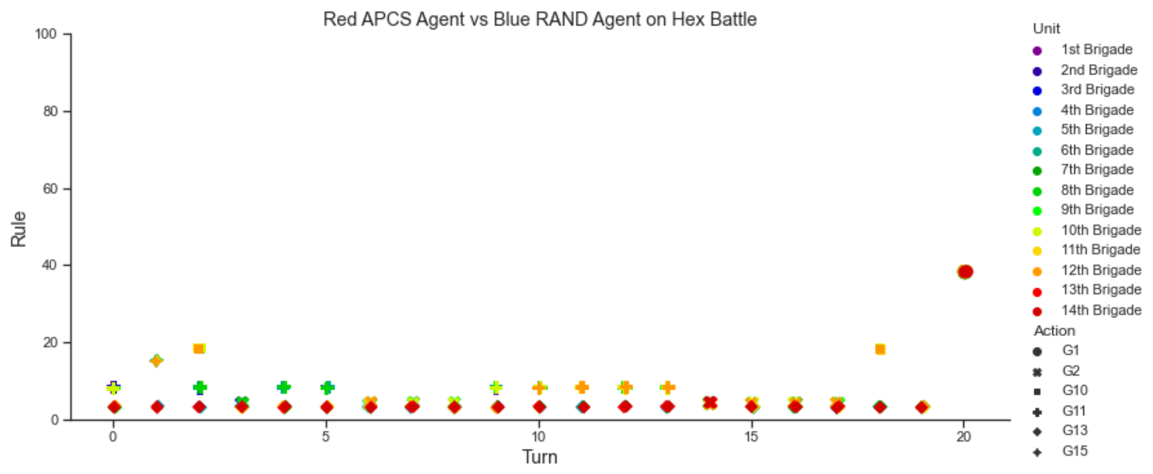


Figure 16: A rule selection chart for red APCS and HtAPCS agents versus a blue RAND agent on Hokkaido.



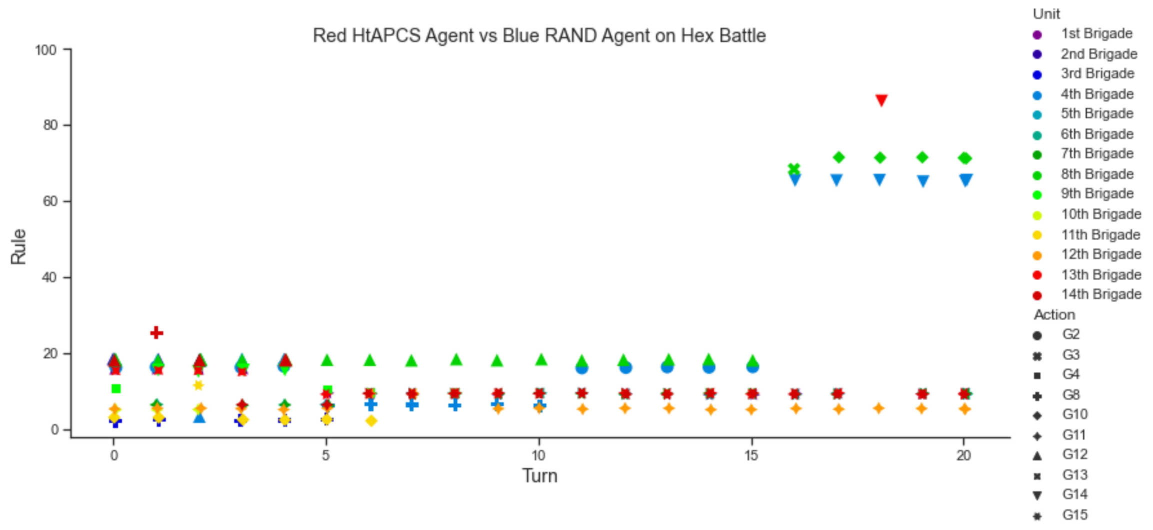
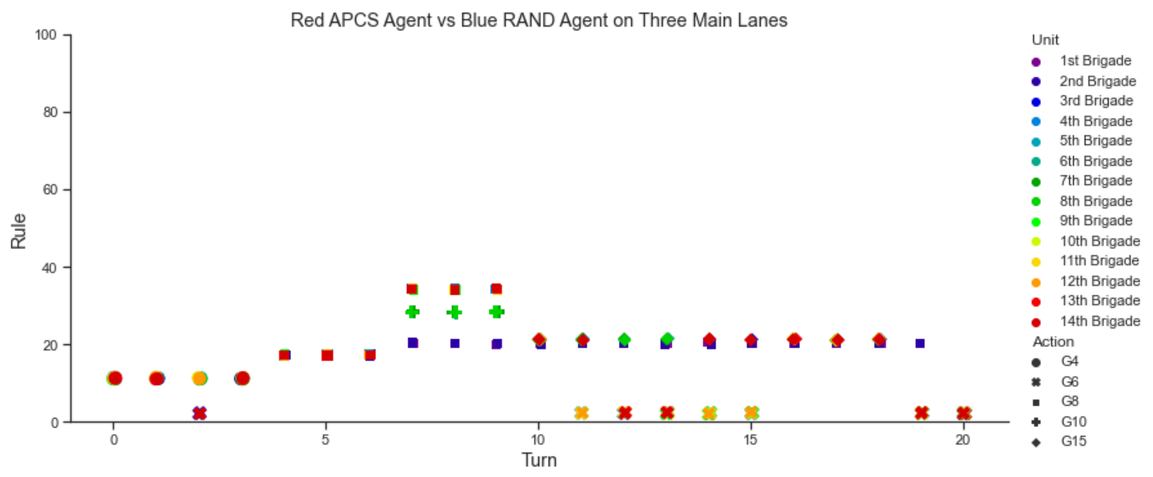


Figure 17: A rule selection chart for red APCS and HtAPCS agents versus a blue RAND agent on Hex Battle.



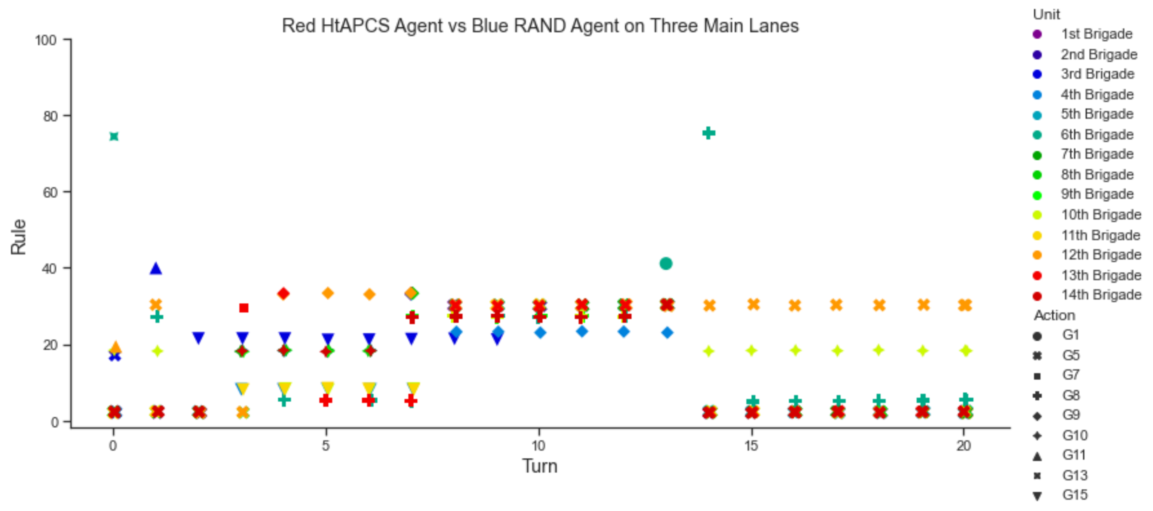
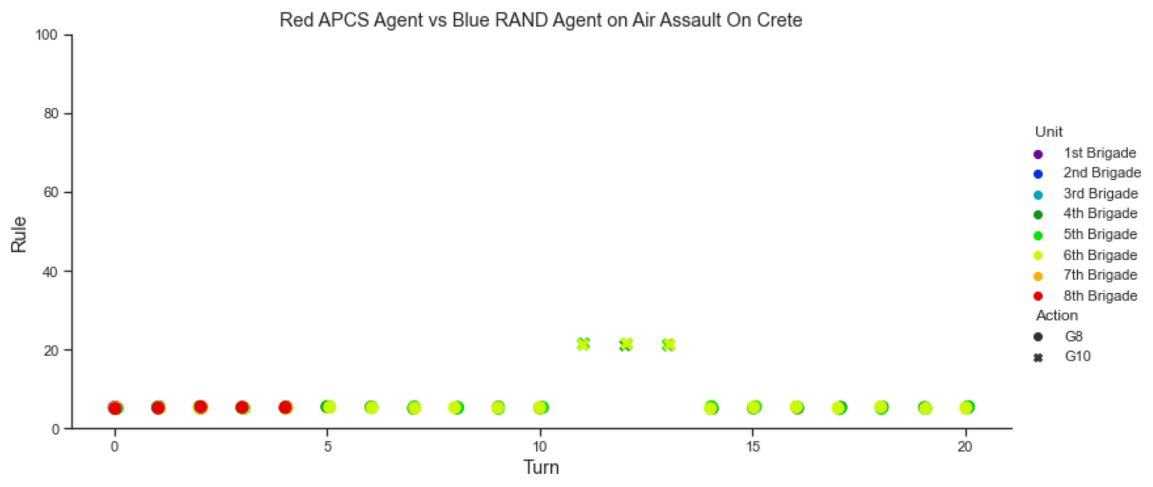


Figure 18: A rule selection chart for red APCS and HtAPCS agents versus a blue RAND agent on Three Main Lanes.



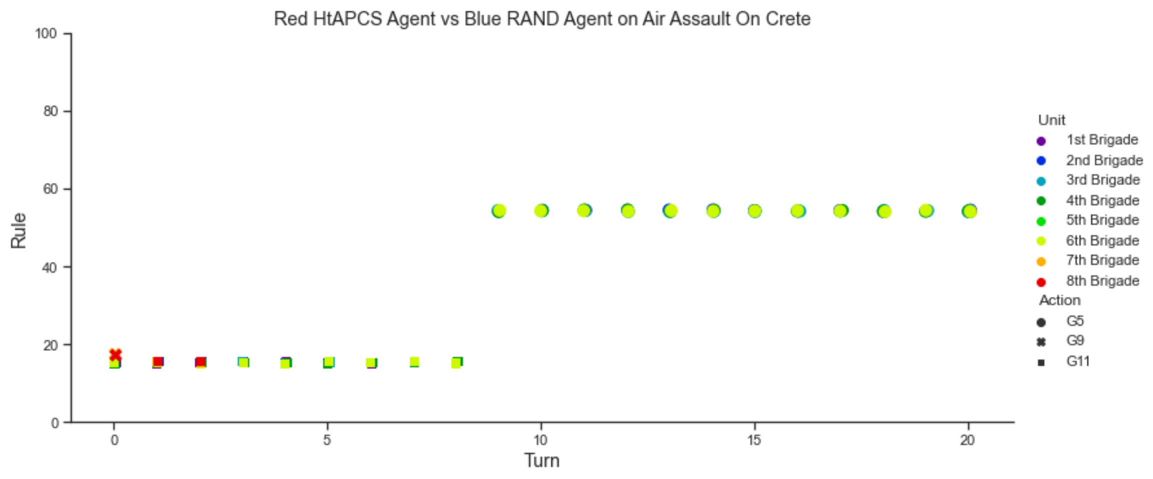
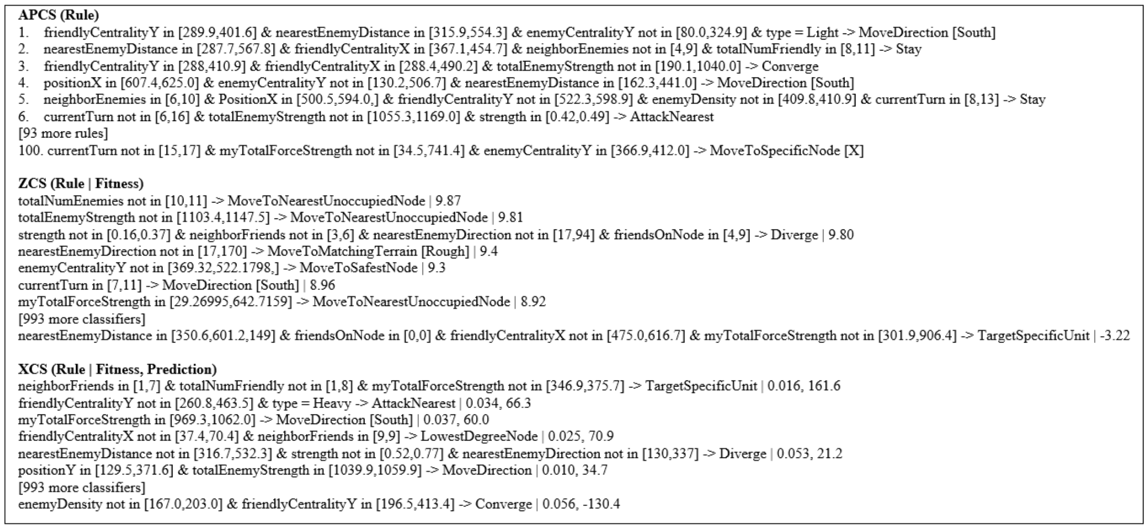


Figure 19: A rule selection chart for red APCS and HtAPCS agents versus a blue RAND agent on Air Assault On Crete.

4.4 Symbolic Analysis

In addition to their contrasting performance, the LCSs' trained rule-sets provide different levels of explainability and replicability. As an example, a sample of the LCSs rule-sets following the tournament on Hokkaido are shown in Figure 20. To replicate the decision-making process of the APCS agent, the human user needs only to read the rule-set as a Fast And Frugal Tree (FFT). If the user has online access to the required information, this process is simple. If the objective is to interpret rather than replicate, the reader can determine valuable heuristics from the rule-set. For example, a simplified interpretation of the first rule is "IF my team is mainly located in the north of the map AND there are no enemies near me AND the enemies are mainly not in the center of the map AND my unit is a light unit THEN Move South". This rule aligns with the heuristics of the game, as it prompts an advancement to a more maneuverable position of the map. They can also infer offline how APCS makes certain associations between state and action. In this particular rule-set, centrality is a frequently occurring condition that prompts a MoveDirection action, and thus the reader can infer that one should consider the centrality of a team before moving a unit a certain direction.

Michigan-style MALCSs do not not possess the explainability of APCS, as a user attempting to interpret or replicate these agents forced to check the condition and track the expected payoff of each classifier in $[M]$. Like APCS, a human user could interpret offline what condition-action pairs prompt good outcomes by looking at the fittest and least fit rules. As an example, if only the first three rules and the last rule in the ZCS rule-set were in $[M]$, the running tally would be {MoveToNearestUnoccupiedNode: 9.84, Diverge, 9.80, TargetSpecificUnit: -3.22}. Given the associated fitnesses, a human can form associations between the current state and the agent's recommended action ranking. An advantage in interpretability not shared with APCS


```
APCS (Rule)
1.   friendlyCentralityY in [289.9,401.6] & nearestEnemyDistance in [315.9,554.3] & enemyCentralityY not in [80.0,324.9] & type = Light -> MoveDirection [South]
2.   nearestEnemyDistance in [287.7,567.8] & friendlyCentralityX in [367.1,454.7] & neighborEnemies not in [4,9] & totalNumFriendly in [8,11] -> Stay
3.   friendlyCentralityY in [288,410.9] & friendlyCentralityX in [288.4,490.2] & totalEnemyStrength not in [190.1,1040.0] -> Converge
4.   positionX in [607.4,625.0] & enemyCentralityY not in [130.2,506.7] & nearestEnemyDistance in [162.3,441.0] -> MoveDirection [South]
5.   neighborEnemies in [6,10] & PositionX in [500.5,594.0,] & friendlyCentralityY not in [522.3,598.9] & enemyDensity not in [409.8,410.9] & currentTurn in [8,13] -> Stay
6.   currentTurn not in [6,16] & totalEnemyStrength not in [1055.3,1169.0] & strength in [0.42,0.49] -> AttackNearest
[93 more rules]
100. currentTurn not in [15,17] & myTotalForceStrength not in [34.5,741.4] & enemyCentralityY in [366.9,412.0] -> MoveToSpecificNode [X]

ZCS (Rule | Fitness)
totalNumEnemies not in [10,11] -> MoveToNearestUnoccupiedNode | 9.87
totalEnemyStrength not in [1103.4,1147.5] -> MoveToNearestUnoccupiedNode | 9.81
strength not in [0.16,0.37] & neighborFriends not in [3,6] & nearestEnemyDirection not in [17,94] & friendsOnNode in [4,9] -> Diverge | 9.80
nearestEnemyDirection not in [17,170] -> MoveToMatchingTerrain [Rough] | 9.4
enemyCentralityY not in [369.32,522.1798,] -> MoveToSafestNode | 9.3
currentTurn in [7,11] -> MoveDirection [South] | 8.96
myTotalForceStrength in [29.26995,642.7159] -> MoveToNearestUnoccupiedNode | 8.92
[993 more classifiers]
nearestEnemyDistance in [350.6,601.2,149] & friendsOnNode in [0,0] & friendlyCentralityX not in [475.0,616.7] & myTotalForceStrength not in [301.9,906.4] -> TargetSpecificUnit | -3.22

XCS (Rule | Fitness, Prediction)
neighborFriends in [1,7] & totalNumFriendly not in [1,8] & myTotalForceStrength not in [346.9,375.7] -> TargetSpecificUnit | 0.016, 161.6
friendlyCentralityY not in [260.8,463.5] & type = Heavy -> AttackNearest | 0.034, 66.3
myTotalForceStrength in [969.3,1062.0] -> MoveDirection [South] | 0.037, 60.0
friendlyCentralityX not in [37.4,70.4] & neighborFriends in [9,9] -> LowestDegreeNode | 0.025, 70.9
nearestEnemyDistance not in [316.7,532.3] & strength not in [0.52,0.77] & nearestEnemyDirection not in [130,337] -> Diverge | 0.053, 21.2
positionY in [129.5,371.6] & totalEnemyStrength in [1039.9,1059.9] -> MoveDirection | 0.010, 34.7
[993 more classifiers]
enemyDensity not in [167.0,203.0] & friendlyCentralityY in [196.5,413.4] -> Converge | 0.056, -130.4
```

Figure 20: A sample of the trained rule-sets on Hokkaido. The six fittest rules and fitnesses are shown, along with the rule that possessed the weakest fitness or predicted payoff.

is the availability of smaller condition clauses, upon which the user can observe the effect of individual conditions. Furthermore, the user may potentially observe what the agent constitutes as bad decisions through negative payoffs. In APCS, the rule-set can be viewed as what the agent defines as the best 100 rules, but the worst rules are not tracked. Lower-ranked rules in APCS imply the advancement to a weaker state but not necessarily a "bad" one. The least fit rule in the ZCS rule-set can be interpreted as "IF there are no enemies near me AND there are no friendly units at my location AND my team is not located in the east portion of the map AND my team's total health is near-depleted or full THEN Attack". This rule makes sense with the nature of the game, as a lone attack without the help of the team would likely result in the destruction of the unit with minimal damage to the opposing team.

The Michigan-style LCSs fall short of APCS when the objective is to replicate the agent. A human user may need to go through all matched rules to completely replicate the outcome of ZCS or XCS. While a human can achieve near-replication by sorting the classifiers in $[M]$ by fitness and noting the action recommended by a sample of the most polar classifiers, it does not guarantee 100% accuracy.

4.5 Towards Interpretability

Despite being shown to be interpretable, LCSs are limited by their ability to filter out irrelevant conditions that consequently joined with relevant conditions. Figure 21 displays the distribution of conditions and actions of the top 10% of ZCS classifiers in the finalized rule-sets. Despite conditions measuring very different aspects of the state-space, no set of rules was over-saturated with a subset of informed conditions, or conditions with high information-gain. This phenomenon suggests that LCSs are incapable of filtering out conditions that do not provide value to the decision-making process, and their inclusion requires unnecessary computation. To further examine this idea, a subsequent experiment was required to determine a MALCS's ability to filter out uniformed conditions. A irrelevant variable was added to the set of available ground conditions. This variable, known as C_T, has a 100% chance of always returning true in a condition clause. Thus, C_T provides no useful information regarding the state-space. Figure 22 demonstrates finalized APCS and ZCS rule-sets with C_T as a rule-set. Despite providing no accurate information about the state-space, the best classifiers in the rule-sets still incorporated them in their condition clauses. This phenomenon conflicts with the established trade-off of keeping classifiers minimal yet accurate. If a human were attempting to recreate the algorithm's decision-making processes, unnecessary computations and condition checks would interfere with the rule-set's interpretability.

There are ways to potentially clean a finalized rule-set generated by LCSs to make it more readable. Wilson proposed Classifier Reduction Algorithm (CRA) [55], which was expanded by Dixon *et al* [56]. This algorithm is a three-step offline learning model which finds the minimal number of classifiers that achieve 100% performance, elimi-

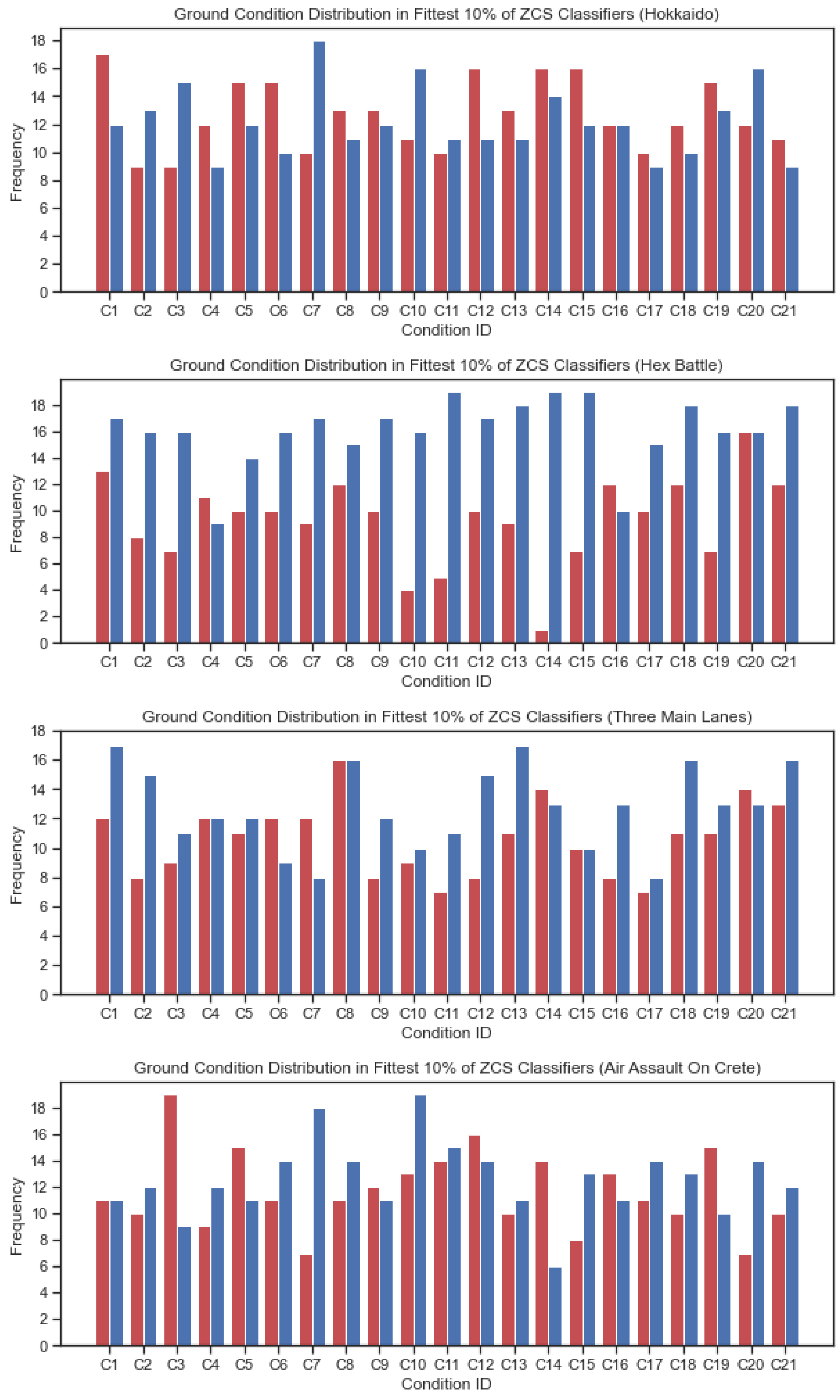


Figure 21: Distribution of conditions in the top 10% of classifiers in the finalized ZCS rule-set.

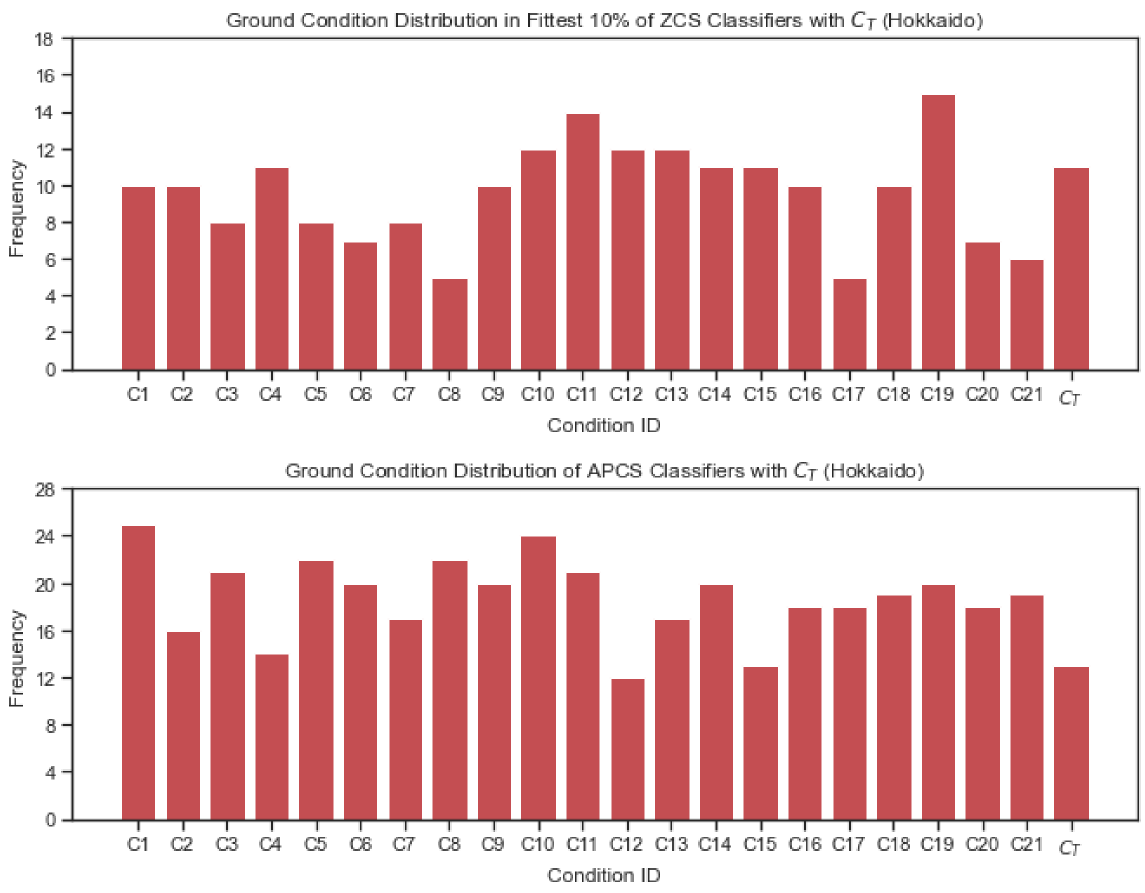


Figure 22: Distribution of conditions in the top 10% of classifiers in the finalized ZCS rule-set.

nates any subset of classifiers that do not improve the overall solution, and discards any unused classifiers. While this scheme could be adapted to other Michigan-style classifiers, a similar algorithm for Pittsburgh-style LCSs is currently unexplored.

A minimizing pipeline is suggested as follows: a Pittsburgh-style agent plays a large number of games (n>100) against a variety of agents. At each step of action selection, the agent tracks which conditions in each classifier return true and which return false. A condition's match rate is defined as the number of times it returns true divided by the total of number times it is checked. In the first step of minimizing, if a single condition's match rate in a classifier is at or below a certain low threshold α (likely near 0%), the entire classifier can be removed, shortening the overall length of the rule-set. In the second step, if a condition's match rate is at or above a certain high threshold β (likely near 100%), the condition can be removed from the finalized list. The exception to this policy is the last classifier, for if the entire condition clause has a match rate of 100%, that classifier is the default action, and all classifiers after it can be removed from the list.

Using this method with $\alpha = 0$ and $\beta = 100$ on the resulting rule-book generated from the red HtAPCS agent playing against all other agents on Air Assault On Crete, 100 ground classifiers averaging 4.06 conditions per classifier was transformed to nine classifiers averaging 3.33 conditions per classifier, and finally 9 classifiers averaging 2.0 conditions per classifier. On Hokkaido, 100 ground classifiers averaging 3.90 conditions per classifier was transformed to 26 classifiers averaging 3.46 conditions per classifier, and finally 26 classifiers averaging 2.80 conditions per classifier. In both cases, there existed no significant difference between the minimized rule-sets and the original rule-sets. However, because the classifiers were tracked through sampling, it is not guaranteed that this method will maintain the same performance, even with non-zero floors or non-hundred ceilings. Another potential method to improve readability is to set an upper-bound to the number of conditions such that APCS can only make decisions based off a restricted set of classifiers and thus would perform poorer if one of the slots required to maintain an accurate representation of the state-space were taken by an uninformed classifier.

V. Conclusions

This chapter provides conclusions to the results presented in Chapter IV and recommendations for future work. Section 5.1 highlights key takeaways from the results. Section 5.2 outlines how this document produced novel research. Section 5.3 covers the limitations of the this document in regards to the experiments. Finally, Section 5.4 outlines the next pressing steps for further research.

5.1 Key Takeaways

The results offer a few key takeaways. First of all, the results demonstrate that Learning Classifier Systems (LCSs) can be effective competitors in Stratagem MIST, showing promise for further LCS experimentation in wargame domains. In the main tournament, the LCS agents outperformed Online Evolutionary Planning (OEP) and the baselines. While OEP persevered in the general case, all LCS agents outperformed the baselines. The most notable achievement is the success of Adaptive Pittsburgh Classifier System (APCS) in an environment that is often saturated with Michigan-style LCSs [7]. Its success as well as its enhanced interpretability may prompt a renaissance of the long-forgotten Pittsburgh-style classifier system.

5.2 Significance of Research

This thesis offered several novel perspectives to the field, to include:

- A direct comparison of action selection strategies for Pittsburgh-style learning, specifically within the APCS algorithm.
- A comparison of heterogeneous and homogeneous control strategies in Multi-Agent Learning Classifier System (MALCS) agents.

- An extension of LCS algorithms to wargames, with a focus on symbolic classifier representation.
- An examination of interpretability of LCSs, with regard to their explainability and potential replicability by a human agent.
- A novel approach to the game of Stratagem MIST, which currently has limited exposure to advanced agents due to its in-development status.

5.3 Limitations

Some limitations of the LCS algorithms occur in the representation of the state-space. One of the main issues with rules consisting of partial representations of the state-space is that in the covering phase and the Genetic Algorithm (GA) phase, an over-saturation of ineffective condition clauses can occur. There are a few methods in literature that aim to correct this problem. Abedini *et al.* developed a feature-ranking system for eXtended Classifier System (XCS) [57]. In this system, a Rule Discovery Probability Vector (RDP) contains a probability associated with each potential condition in $C = \{c_1, c_2, ..., c_n\}$, with all the probabilities adding up to 1.0. When a condition is generated in covering or through the GA component, the conditions with the higher probability in the RDP are more likely to be chosen. Abedini *et al.* utilized supervised learning methods such as Information Gain and Entropy to determine the distribution of the RDP. To the best of the author's knowledge, this phenomenon has not been extended to Pittsburgh-style classifier systems and/or reinforcement learning domains. The need for a feature-ranking system is amplified by the incorporation of heuristic-based actions. Available actions are formed from the game's default *Move Unit A to Node X* action in addition to heuristic-based actions as chosen by the algorithm implementer. While the premise behind implementing non-default actions is that the GA would filter out non-relevant actions, there exists a possibility that these actions add a layer of overhead to the algorithm, leading to slower convergence. A RDP that diminishes actions known to lead to inferior states would allow an implementer to experiment with new heuristic-based actions without consequence of slowing down the GA.

Another limitation was the simplicity of the multi-agent systems in question. In the homogeneous systems, the agents derive from the same rule-set and therefore "think" the same in the absence of unit-specific conditions. Therefore, it makes it easier for clustering to occur compared to a heterogeneous system, even with homogeneous seeding. There likely exist enhancements to the multi-agent system that would aid the algorithms, such as rule-sharing across units, a messaging system across units to include the proposed actions of fellow units as a condition, and a reward system that combines greedy individual rewards with a collective team reward.

5.4 Future Work

This thesis paves paths for future work, to include:

- Further analysis on other LCS implementations. The list of LCS algorithms compared in this paper is by no means exhaustive. In [8], R. Urbanowicz lists 84 LCS implementations. The implementations in this thesis were not chosen by their novelty but by their frequency in literature. While the results of the three LCS algorithms showed that LCSs can perform well in a wargame environment, a pressing step would be to test out more modern implementations to find a best LCS algorithm for the domain.

- Exploration of Control Strategies. While LCS literature is heavily concentrated on single-agent implementations, there is much more work to be done with control strategies. The multi-agent LCSs all used a decentralized approach in which units moved independently of each other and had no knowledge of the planned actions of other units. A centralized agent that acts as a "commander" may be more in line with how wargames operate, as real-life military units would follow the orders of a superior rather than operate entirely independently or only with a set of static predefined orders.

- Extension to other wargames. This thesis showed that symbolic game-playing methods have a place in wargames through its success in Stratagem MIST. An extension to other graph-based, multi-action games could test if the results are domain-specific or can be generalizable.
- A revisit to a release version. At time of writing, Stratagem MIST is still in-development. Thus, game features and scenarios added after Release 4 are not present in this document. While the game has potential to generate complex environmental configuration in terms of graph layout, unit placement, and metric/objective setups, the domains were limited to just four scenarios. As more scenarios and features are released, a pressing step would be to verify that the algorithms explored in this document maintain their effectiveness. This is especially true with the scenario-general experimental configuration, as just four scenarios could cause over-fitting to small sample size of environments rather than reflect an agent's ability to generalize on an unseen scenario.
- LCS/human pairing. The results showed that symbolic methods are an effective way to play Stratagem MIST compared to the baseline agents and the state-of-the-art OEP algorithm. One of the distinct features of LCSs is the end product of a human-readable rule-set upon which a human player can follow to play the game effectively. A further step is to test whether or not a human given a rule-

set generated by an algorithm such as APCS could effectively learn the ideal rules and strategies quicker than a control group. Furthermore, the APCS with Fast And Frugal Tree (FFT) action selection could even act as a testing ground for human players to evaluate their own rules. As a preliminary experiment for future work, the author seeded hand-crafted rules immune to mutation in each individual classifier. The hand-crafted rules had to be included in the final rule-set, but could be swapped with an existing rule. After a few generations, the rules that were expected to be representative of a heuristic for Stratagem MIST were cycled to the top of the rule-sets in most generations, while the assumed weak-rules were pushed to the bottom of the decision list. While this example requires formal experimentation, it shows that the extent how LCS (specifically APCS) can teach humans has much to be explored.

Appendix A. Results Tables

Table A.1: Hokkaido Win Rates. The win rate is read as ROW/COL.

Red/Blue	(B) RAND	(B) RM-MAAB	(B) RM-A2G	(B) OEP	(B) ZCS	(B) XCS	(B) APCS	(B) Ht-ZCS	(B) Ht-XCS	(B) Ht-APCS
(R) RAND	39.4/60.6	0/100	0/100	0/100	0/100	0/100	0/100	0/100	6.7/93.3	0/100
(R) RM-MAAB	99.2/0.8	0/100	0/100	33.8/66.2	0/100	0/100	0/100	0/100	0/100	0/100
(R) RM-A2G	99.2/0.8	0/100	0/100	33/67	0/100	0/100	0/100	0/100	0/100	0/100
(R) OEP	100/0	55.8/44.2	57.4/42.6	37.1/62.9	0.3/99.7	8.1/91.9	0.2/99.8	30.3/69.7	100/0	8.1/91.9
(R) ZCS	100/0	100/0	100/0	91.7/8.3	99.8/0.2	0/100	0/100	100/0	100/0	100/0
(R) XCS	100/0	100/0	100/0	61.3/38.7	0/100	0/100	0/100	0/100	100/0	0/100
(R) APCS	99.5/0.5	100/0	100/0	97.9/2.1	100/0	86.3/13.7	0/100	37.8/62.2	100/0	100/0
(R) Ht-ZCS	100/0	100/0	100/0	32.6/67.4	0/100	0/100	0/100	0/100	100/0	0/100
(R) Ht-XCS	99.8/0.2	0/100	0/100	11.5/88.5	0/100	0/100	0/100	0/100	99/1	0/100
(R) Ht-APCS	99.7/0.3	100/0	100/0	97.3/2.7	100/0	100/0	0/100	100/0	100/0	100/0

Table A.2: The final scores for the Hokkaido Scenario.

	(B) RAND	(B) RM-MAAB	(B) RM-A2G	(B) OEP	(B) ZCS	(B) XCS	(B) APCS	(B) Ht-ZCS	(B) Ht-XCS	(B) Ht-APCS	Avg
(R) RAND	-0.080	-1.458	-1.454	-1.315	-1.647	-1.595	-1.631	-1.331	-0.471	-1.000	-1.198
(R) RM-MAAB	1.089	-0.566	-0.566	-0.300	-1.817	-1.735	-1.788	-1.760	-0.523	-0.897	-0.886
(R) RM-A2G	1.076	-0.567	-0.566	-0.316	-1.817	-1.735	-1.783	-1.760	-0.528	-0.897	-0.889
(R) OEP	1.169	-0.078	-0.066	-0.128	-1.011	-0.756	-1.205	-0.326	0.926	-0.528	-0.200
(R) ZCS	1.596	1.175	1.175	0.651	0.220	-0.080	-0.940	0.024	1.517	0.375	0.571
(R) XCS	1.446	1.624	1.624	0.066	-0.861	-0.302	-1.323	-0.507	0.220	0.000	0.199
(R) APCS	1.142	1.080	1.080	1.164	0.338	0.162	-1.504	-0.110	0.983	1.344	0.568
(R) Ht-ZCS	1.329	0.430	0.431	-0.023	-0.324	-1.045	-0.028	-0.265	1.456	-0.552	0.141
(R) Ht-XCS	0.868	-1.612	-1.558	-0.455	-0.966	-0.613	-1.617	-1.433	0.047	-0.396	-0.773
(R) Ht-APCS	1.131	0.998	0.998	0.820	0.249	0.348	-0.418	0.117	0.329	0.563	0.514
Avg	1.077	0.103	0.110	0.016	-0.763	-0.735	-1.224	-0.735	0.396	-0.199	

Table A.3: The win rates for the Hex Battle Scenario. The win rate is read as ROW/COL.

Red/Blue	**(B) RAND**	**(B) RM-MAAB**	**(B) RM-A2G**	**(B) OEP**	**(B) ZCS**	**(B) XCS**	**(B) APCS**	**(B) Ht-ZCS**	**(B) Ht-XCS**	**(B) Ht-APCS**
(R) RAND	52.4/47.6	9.3/90.7	10.5/89.5	0/100	0/100	9.3/90.7	0/100	0/100	0/100	0/100
(R) RM-MAAB	97.8/2.2	100/0	100/0	93.2/6.8	0/100	0/100	0/100	0/100	100/0	0/100
(R) RM-A2G	97.7/2.3	100/0	100/0	94.2/5.8	0/100	0/100	0/100	0/100	100/0	0/100
(R) OEP	100/0	6.7/93.3	6.9/93.1	49.9/50.1	11.7/88.3	99.6/0.4	0.1/99.9	2.3/97.7	47.5/52.5	0.2/99.8
(R) ZCS	100/0	100/0	100/0	100/0	100/0	100/0	74.6/25.4	100/0	100/0	0/100
(R) XCS	99.3/0.7	100/0	100/0	25.1/74.9	0/100	0/100	100/0	0/100	100/0	0/100
(R) APCS	97/3	99.9/0.1	100/0	97/3	67.6/32.4	100/0	0.1/99.9	0/100	36/64	96.1/3.9
(R) Ht-ZCS	100/0	100/0	100/0	99.8/0.2	100/0	100/0	33.8/66.2	0/100	100/0	0/100
(R) Ht-XCS	100/0	87.2/12.8	27.2/72.8	88/12	100/0	100/0	0/100	54/46	100/0	0/100
(R) Ht-APCS	100/0	94.5/5.5	95/5	99.9/0.1	0/100	100/0	0/100	100/0	99.9/0.1	89.5/10.5

Table A.4: The final scores for the Hex Battle Scenario.

	(B) RAND	**(B) RM-MAAB**	**(B) RM-A2G**	**(B) OEP**	**(B) ZCS**	**(B) XCS**	**(B) APCS**	**(B) Ht-ZCS**	**(B) Ht-XCS**	**(B) Ht-APCS**	**Avg**
(R) RAND	0.098	-0.702	-0.659	-1.327	-1.586	-1.207	-1.673	-2.127	-1.391	-1.466	-1.204
(R) RM-MAAB	0.886	0.154	0.154	0.895	-1.855	-1.701	-0.535	-0.501	1.093	-1.803	-0.321
(R) RM-A2G	0.885	0.154	0.154	0.917	-1.855	-1.696	-0.535	-0.502	1.151	-1.803	-0.313
(R) OEP	1.620	-0.874	-0.872	0.051	-0.577	1.462	-1.539	-0.948	-0.016	-1.372	-0.306
(R) ZCS	2.629	0.971	0.971	1.376	0.488	2.407	0.124	0.524	1.775	-0.402	1.086
(R) XCS	1.408	1.361	1.361	-0.300	-0.688	-1.090	1.801	-1.268	0.607	-0.828	0.236
(R) APCS	1.069	0.230	0.229	1.068	0.706	1.931	-0.729	-0.475	-0.103	0.405	0.433
(R) Ht-ZCS	1.235	1.334	1.334	1.571	1.403	1.147	-0.100	-0.561	1.812	-0.727	0.845
(R) Ht-XCS	1.997	0.245	-0.279	0.745	0.778	1.296	-1.942	0.050	1.028	-0.919	0.300
(R) Ht-APCS	1.820	0.202	0.207	1.756	-0.913	0.300	-1.224	1.619	1.268	0.926	0.596
Avg	1.365	0.307	0.260	0.675	-0.410	0.285	-0.635	-0.419	0.722	-0.799	

Table A.5: The win rates for the Three Main Lanes scenario. The win rate is read as ROW/COL.

Red/Blue	**(B) RAND**	**(B) RM-MAAB**	**(B) RM-A2G**	**(B) OEP**	**(B) ZCS**	**(B) XCS**	**(B) APCS**	**(B) Ht-ZCS**	**(B) Ht-XCS**	**(B) Ht-APCS**
(R) RAND	52.3/47.7	100/0	100/0	8.9/91.1	3/97	68.7/31.3	15.2/84.8	32.1/67.9	0.3/99.7	1.8/98.2
(R) RM-MAAB	0/100	0/100	0/100	0/100	0/100	0/100	0/100	0/100	0/100	0/100
(R) RM-A2G	0/100	0/100	0/100	0/100	0/100	0/100	0/100	0/100	0/100	0/100
(R) OEP	95.3/4.7	100/0	100/0	63.1/36.9	13.8/86.2	96.6/3.4	3/97	44.1/55.9	3.9/96.1	6.5/93.5
(R) ZCS	95.6/4.4	100/0	100/0	77.2/22.8	100/0	0/100	100/0	0/100	100/0	100/0
(R) XCS	10.8/89.2	100/0	100/0	0/100	0/100	0/100	0/100	0/100	0/100	0/100
(R) APCS	94.2/5.8	100/0	100/0	91.5/8.5	67/33	0/100	100/0	4.2/95.8	78.7/21.3	99.7/0.3
(R) Ht-ZCS	20.9/79.1	100/0	100/0	36.9/63.1	100/0	100/0	100/0	65.3/34.7	89.9/10.1	0.5/99.5
(R) Ht-XCS	99.5/0.5	100/0	100/0	92.4/7.6	70.6/29.4	0/100	0/100	99.1/0.9	86.2/13.8	100/0
(R) Ht-APCS	98.8/1.2	100/0	100/0	93.5/6.5	0/100	0/100	82.1/17.9	0/100	100/0	100/0

Table A.6: The final scores for the Three Main Lanes scenario. Unlike the other scenarios, a higher score does not necessarily equate to a win due to the presence of an objective.

	(B) RAND	**(B) RM-MAAB**	**(B) RM-A2G**	**(B) OEP**	**(B) ZCS**	**(B) XCS**	**(B) APCS**	**(B) Ht-ZCS**	**(B) Ht-XCS**	**(B) Ht-APCS**	**Avg**
(R) RAND	0.018	0.557	0.553	-0.268	-0.414	0.074	-0.202	-0.070	-0.468	-0.491	-0.071
(R) RM-MAAB	-0.543	0.000	0.000	-0.431	-0.344	-0.004	-0.066	-0.013	-0.328	-0.438	-0.217
(R) RM-A2G	-0.549	0.000	0.000	-0.428	-0.344	-0.004	-0.066	-0.013	-0.328	-0.435	-0.217
(R) OEP	0.335	0.474	0.474	0.058	-0.195	0.171	-0.328	-0.006	-0.414	-0.463	0.011
(R) ZCS	0.187	0.474	0.474	0.071	0.175	-0.212	0.516	-0.148	0.368	0.432	0.234
(R) XCS	-0.662	0.008	0.008	-0.863	-0.657	-0.192	-0.210	-0.075	-0.569	-0.676	-0.389
(R) APCS	0.272	0.476	0.476	0.254	0.043	-0.150	0.578	-0.112	0.155	0.346	0.234
(R) Ht-ZCS	-0.125	0.004	0.004	-0.061	0.189	0.004	0.071	0.025	0.103	-0.238	-0.002
(R) Ht-XCS	0.374	0.269	0.270	0.318	0.079	-0.118	-0.817	0.217	0.390	0.581	0.156
(R) Ht-APCS	0.461	0.591	0.591	0.356	-0.144	-0.100	0.078	-0.107	0.639	0.406	0.277
Avg	-0.023	0.285	0.285	-0.099	-0.161	-0.053	-0.045	-0.030	-0.045	-0.098	

Table A.7: Air Assault On Crete Win Rates. The win rate is read as ROW/COL.

Red/Blue	(B) RAND	(B) RM-MAAB	(B) RM-A2G	(B) OEP	(B) ZCS	(B) XCS	(B) APCS	(B) Ht-ZCS	(B) Ht-XCS	(B) Ht-APCS
(R) RAND	4.9/95.1	1.1/98.9	1/99	0/100	0/100	0/100	0/100	0.8/99.2	0/100	0/100
(R) RM-MAAB	1/99	0/100	0/100	0/100	0/100	0/100	0/100	0/100	0/100	0/100
(R) RM-A2G	1.1/98.9	0/100	0/100	0.1/99.9	0/100	0/100	0/100	0/100	0/100	0/100
(R) OEP	97.4/2.6	2.6/97.4	3.3/96.7	12.5/87.5	0.4/99.6	3.5/96.5	0/100	5.7/94.3	3.8/96.2	0/100
(R) ZCS	99.1/0.9	100/0	100/0	5.6/94.4	0/100	6.2/93.8	0/100	25.6/74.4	0/100	0/100
(R) XCS	0.1/99.9	100/0	100/0	0/100	0/100	0/100	0/100	0/100	0/100	0/100
(R) APCS	93.9/6.1	100/0	100/0	93.4/6.6	77.8/22.2	38.9/61.1	0/100	100/0	0/100	0/100
(R) Ht-ZCS	48/52	99.2/0.8	99.5/0.5	2/98	0/100	0/100	0/100	0/100	0/100	0/100
(R) Ht-XCS	33.3/66.7	0/100	0/100	0/100	0/100	0/100	0/100	0/100	0/100	0/100
(R) Ht-APCS	100/0	100/0	100/0	54.7/45.3	98.4/1.6	0/100	0/100	0/100	0.9/99.1	0/100

Table A.8: The final scores for the Air Assault On Crete Scenario.

	(B) RAND	(B) RM-MAAB	(B) RM-A2G	(B) OEP	(B) ZCS	(B) XCS	(B) APCS	(B) Ht-ZCS	(B) Ht-XCS	(B) Ht-APCS	Avg
(R) RAND	-0.206	-0.540	-0.542	-0.628	-0.696	-0.707	-0.739	0.001	-0.678	-0.821	-0.556
(R) RM-MAAB	-0.656	-0.464	-0.464	-0.699	-0.810	-0.861	-0.693	-0.761	-0.766	-0.753	-0.693
(R) RM-A2G	-0.648	-0.463	-0.464	-0.698	-0.810	-0.861	-0.692	-0.762	-0.766	-0.753	-0.692
(R) OEP	0.310	-0.525	-0.509	-0.230	-0.094	-0.519	-0.697	0.008	-0.562	-0.733	-0.355
(R) ZCS	0.370	0.643	0.643	-0.260	-0.196	-0.761	-0.647	-0.002	-0.770	-0.633	-0.161
(R) XCS	-0.066	0.153	0.295	-0.383	-0.243	-0.722	-0.754	-0.294	-0.385	-0.827	-0.322
(R) APCS	0.271	-0.747	-0.747	0.292	0.005	-0.157	-0.627	0.536	-0.557	-0.411	-0.214
(R) Ht-ZCS	-0.024	0.282	0.284	-0.450	-0.452	-0.869	-0.639	-0.108	-0.712	-0.686	-0.338
(R) Ht-XCS	-0.369	-0.872	-0.872	-0.854	-0.862	-0.824	-0.872	-0.165	-0.790	-0.651	-0.713
(R) Ht-APCS	0.561	0.210	0.211	0.040	0.020	-0.818	-0.462	-0.007	-0.331	-0.668	-0.124
Avg	-0.046	-0.232	-0.217	-0.387	-0.414	-0.710	-0.682	-0.155	-0.632	-0.694	

Appendix B. Supporting Tables

Table B.1: Absolute Conditions for the LCS Agents.

ID	**Name**	**Operator**	**General/ Specific**	**Description**
C1	Current Turn	Interval	General	Determines if the current turn in the game is in or out of the desired interval.
C2	Enemy Density	Interval	General	Calculates the average Euclidean distance of the point of enemy centrality to each enemy unit and determines if it is in or out of the desired interval.
C3	Total Enemy Strength	Interval	General	Determines if the total combined strength of enemy ground units is in or out of the desired interval.
C4	Total Force Strength	Interval	General	Determines if the total combined strength of friendly ground units is in or out of the desired interval.
C5	Total # Enemies	Interval	General	Determines if the total number of enemy ground units is in or out of the desired interval.
C6	Total # Friendly Units	Interval	General	Determines if the total number of friendly ground units is in or out of the desired interval.
C7	Metrics Progress	Interval	General	Determines if a metric's value is in or out of the desired interval
C8	Enemy Force Centrality X	Interval	Specific	Determines if the total centrality of enemy forces (x-axis) is in or out of the desired interval.
C9	Enemy Force Centrality Y	Interval	Specific	Determines if the total centrality of enemy forces (y-axis) is in or out of the desired interval.
C10	Force Centrality X	Interval	Specific	Determines if the total centrality of friendly forces (x-axis) is in or out of the desired interval.
C11	Force Centrality Y	Interval	Specific	Determines if the total centrality of friendly forces (y-axis) is in or out of the desired interval.

Table B.2: Available Relative Conditions for the LCS Agents.

ID	**Name**	**Operator**	**General/ Specific**	**Description**
C12	Enemy Units on Neighboring Nodes	Interval	General	Determines if the number of enemy units on the neighboring nodes of the closest node to the ground unit is in or out of the desired interval.
C13	Friendly Units on Neighboring Nodes	Interval	General	Determines if the number of friendly units on the neighboring nodes of the closest node to the ground unit is in or out of the desired interval.
C14	Friendly Units on Closest Node	Interval	General	Determines if the number of friendly units on the closest node to the ground unit is in or out of the desired interval.
C15	Strength	Interval	General	Determines if the strength of the unit (as a % of starting strength) is in or out of the desired interval.
C16	Unit Type	Membership	General	Determines if a unit has membership of a subset of the five unit types.
C17	Nearest Enemy Distance	Interval	General	Determines if the Euclidean distance to the nearest enemy is in or out of the desired interval.
C18	Nearest Enemy Direction	Interval	Specific	Determines if the degree to the nearest enemy is in or out of the desired interval.
C19	Unit Position X	Interval	Specific	Determines if the x position of the unit is in or out of the desired interval.
C20	Unity Position Y	Interval	Specific	Determines if the y position of the unit is in or out of the desired interval.
C21	Unit Name	Membership	Specific	Determines if a unit's unique ID matches the desired value.

Table B.3: Available Ground Actions for the LCS Agents. It is important to note that within each action exist certain action modifiers that can further distinguish actions from another.

	Name	**General/ Specific**	**Description**
G1	Converge	General	Combines forces with nearby units.
G2	Diverge	General	Moves in separate directions to a neighboring node to cover more ground.
G3	Double Diverge	General	Diverges to a neighboring node, and diverge again.
G4	Attack Heaviest Enemy Node	General	Moves to the node with the heaviest enemy strength.
G5	Attack Lightest Enemy Node	General	Moves to the node with the lightest enemy strength that is greater than zero.
G6	Attack Nearest Enemy Node	General	Moves to the nearest node with an enenmy on it.
G7	Move to Unoccupied Node	General	Moves to the nearest node that is not occupied by a team.
G8	Stay	General	Stays in current position.
G9	Move to Highest Degree Node	General	Move to node within specified radius with the highest vertex degree.
G10	Move to Lowest Degree Node	General	Move to node within specified radius with the lowest vertex degree.
G11	Move to Nearest Terrain	General	Move to the nearest node with terrain of specified type.
G12	Move Direction	General	Move to the node for which the angle between the starting position and the node's position is closest to the specified target value.
G13	Evade	General	Move to a neighboring node that is furthest from any enemy.
G14	Move to Specific Node	Specific	Attempts to move to a specific node.
G15	Target Units	Specific	Attempts to move to the location of a unit from a target list. If the target has been eliminated, move on to the next target.

Table B.4: Available Air Actions for the LCS Agents. It is important to note that within each action exist certain action modifiers that can further distinguish actions from another.

	Name	**General/ Specific**	**Description**
A1	Randomly-Generated Unit-Specific Air Plan	Specific	Uses a randomly-generated Air Plan that sets CAS priority, CA priority, and OCA/DCA priority. It then apportions Strike aircraft, Multi-role aircraft, Counter-Air aircraft, and SSM aircraft based on unit-specific target values.
A2	GA-optimized Specific Air Plan	Specific	Uses a pre-determined rule-set optimized by a GA to include specific units.
A3	Murder All Air Bases Air Plan	Both	Uses an Air Plan that targets enemy Air Base Units exclusively at the cost of everything else. If the scenario is known, a specific air base is targeted. Otherwise, a single one is randomly selected.
A4	Air To Ground Focus Air Plan	Both	Uses an Air Plan that takes all of its aircraft and sets it on an Air Interdiction task. For each turn, a single ground unit will be targeted by all aircraft in the Air Plan. If the scenario is known, a specific named unit is targeted. Otherwise, one is randomly selected.
A5	Randomly-Generated Unit-General Air Plan	General	Uses a randomly-generated Air Plan that sets CAS priority, CA priority, and OCA/DCA priority. Specific air units are not apportioned.
A6	GA-optimized General Air Plan	General	Uses a pre-determined rule-set optimized by a GA that excludes naming specific units.

Table B.5: Information regarding Stratagem MIST's default scenarios. For the Objectives & Metrics column, the Enemy Forces Degraded (EFD), Self-Forces Preserved (SFD), Percent Territory Value Consecutively (PTVC), and Percent Territory Value Enemy Consecutively (PTVEC) metrics were used. The Territory Value Held Consecutively Objective (TVHC) is the only objective used in the default scenario.

	Nodes	**Edges**	**Air Reg-ions**	**Ground Units (Red/Blue)**	**Air Units (Red/Blue)**	**Objectives & Metrics**	**Max Turn Count**
Hokkaido	40	52	3	4/2 Light Infantry 2/2 Heavy Infantry 1/0 Mech Infantry 1/5 Light Armor 2/1 Heavy Armor	1/1 F-16 1/1 F-22 1/1 A-10	EFD SFP PTVC PTVEC	20
Hex Battle	94	230	6	2/2 Light Infantry 1/1 Heavy Infantry 3/3 Mech Infantry 4/4 Light Armor 4/4 Heavy Armor	2/2 F-16 1/1 F-22 1/1 A-10	EFD SFP PTVC PTVEC	20
Three Main Lanes	41	54	5	2/2 Light Infantry 3/3 Heavy Infantry 3/3 Mech Infantry 4/4 Light Armor 2/2 Heavy Armor	1/1 F-16 1/1 F-22 1/1 A-10	EFD TVHC	20
Air Assault On Crete	20	25	5	5/2 Light Infantry 2/2 Heavy Infantry 1/1 Light Armor 0/2 Heavy Armor	1/1 F-16 1/1 F-22 1/1 A-10	EFD PTVC	20

Bibliography

1. Julian Togelius, Jesper Juul, Geoffrey Long, William Uricchio, and Mia Consalvo. *In the Beginning of AI, There Were Games*, pages 1–10. 2018.

2. A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM J. Res. Dev.*, 3(3), July 1959.

3. Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M. Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. The 2014 General Video Game Playing Competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):229–243, 2016.

4. David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–, October 2017.

5. Claire Nicodeme. Build confidence and acceptance of AI-based decision support systems - Explainable and liable AI. In *2020 13th International Conference on Human System Interaction (HSI)*, pages 20–23, 2020.

6. Hsinchun Chen, P. Buntin Rinde, Linlin She, S. Sutjahjo, C. Sommer, and D. Neely. Expert prediction, symbolic learning, and neural networks. An experiment on greyhound racing. *IEEE Expert*, 9(6):21–27, 1994.

7. Kamran Shafi and Hussein A. Abbass. A Survey of Learning Classifier Systems in Games [Review Article]. *IEEE Computational Intelligence Magazine*, 12(1):42–55, 2017.

8. Ryan Urbanowicz and Jason Moore. Learning Classifier Systems: A Complete Introduction, Review, and Roadmap. *Journal of Artificial Evolution and Applications*, 2009, 09 2009.

9. Ryan Small and Clare Bates Congdon. Agent Smith: Towards an evolutionary rule-based agent for interactive dynamic games. In *2009 IEEE Congress on Evolutionary Computation*, pages 660–666. IEEE, 2009.

10. Stephen F. Smith. Flexible Learning of Problem Solving Heuristics through Adaptive Search. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'83, page 422–425, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.

11. John Grefenstette and Robert Daley. Methods for competitive and cooperative co-evolution. In *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*, pages 45–50, 1996.

12. Joint Chiefs of Staff. Joint Publication 5-0: Joint Planning, 2020.

13. Jeffrey M. Erickson and Garrett D. Heath. Closing the gap between simulations for training and wargaming. In *2019 Winter Simulation Conference (WSC)*, pages 2515–2523, 2019.

14. Roger C Mason. Wargaming: its history and future. *The International Journal of Intelligence, Security, and Public Affairs*, 20(2):77–101, 2018.

15. Matthew B Caffrey. *On wargaming: How wargames have shaped history and how they may shape the future*, volume 43. Naval War College Press, 2019.

16. Danielle C. Tarraf, J. Michael Gilmore, D. Sean Barnett, Scott Boston, David R. Frelinger, Daniel Gonzales, Alexander C. Hou, and Peter Whitehead. *"An Exper-*

iment in Tactical Wargaming with Platforms Enabled by Artificial Intelligence". RAND Corporation, Santa Monica, CA, 2020.

17. Fernando Fradique Duarte, Nuno Lau, Artur Pereira, and Luis Paulo Reis. A Survey of Planning and Learning in Games. *Applied Sciences*, 10(13), 2020.

18. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, USA, 3rd edition, 2009.

19. Fernando de Mesentier Silva, Julian Togelius, Frank Lantz, and Andy Nealen. Generating Novice Heuristics for Post-Flop Poker. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2018.

20. Ronald L Rivest. Learning decision lists. *Machine learning*, 2(3):229–246, 1987.

21. Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

22. A. Sally Hassan and M. A. Muhammad Rafie. A survey of Game Theory using Evolutionary Algorithms. In *2010 International Symposium on Information Technology*, volume 3, pages 1319–1325, 2010.

23. Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, USA, 1998.

24. Mitchell A. Potter and Kenneth A. De Jong. "A cooperative coevolutionary approach to function optimization". In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature — PPSN III*, pages 249–257, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

25. John H Holland and Judith S Reitman. Cognitive systems based on adaptive algorithms. In *Pattern-directed inference systems*, pages 313–329. Elsevier, 1978.

26. John H Holmes, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W Wilson. Learning classifier systems: New models, successful applications. *Information Processing Letters*, 82(1):23–30, 2002.

27. Ryan John. Urbanowicz. *The detection and characterization of epistasis and heterogeneity: a Learning Classifier System approach.* PhD thesis, Dartmouth College, 2012.

28. Stephen Frederick Smith. *A learning system based on genetic adaptive algorithms.* University of Pittsburgh, 1980.

29. Stewart W. Wilson. ZCS: A Zeroth Level Classifier System. *Evol. Comput.*, 2(1):1–18, March 1994.

30. David E. Goldberg. Genetic Algorithms in Search Optimization and Machine Learning. 1988.

31. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 06 1995.

32. Will Browne, Dan Scott, and Charalambos Ioannides. *Abstraction for Genetics-Based Reinforcement Learning.* 01 2008.

33. Martin V. Butz. Learning Classifier Systems. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '10, page 2331–2352, New York, NY, USA, 2010. Association for Computing Machinery.

34. Jaume Bacardit, Ester Bernadó-Mansilla, and Martin Butz. Learning Classifier Systems: Looking Back and Glimpsing Ahead. volume 4998, pages 1–21, 01 2007.

35. Ester Bernadó, Xavierllo, and Josep-Maria Garrell. XCS and GALE: A comparative study of two learning classifier systems and six other learning algorithms on classification tasks. *LNAI*, 2321, 01 2002.

36. Jaume Bacardit and Martin Butz. Data Mining in Learning Classifier Systems: Comparing XCS with GAssist. volume 4399, pages 282–290, 01 2005.

37. Gilles Enée and Mathias Peroumalnaïk. Adapted Pittsburgh Classifier System: Building Accurate Strategies in Non Markovian Environments. In *Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '08, page 2001â€“2008, New York, NY, USA, 2008. Association for Computing Machinery.

38. David G. Nidorf, Luigi Barone, and Tim French. A comparative study of NEAT and XCS in Robocode. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

39. Ryan Urbanowicz and Jason Moore. ExSTraCS 2.0: Description and Evaluation of a Scalable Learning Classifier System. *Evolutionary Intelligence*, 8, 09 2015.

40. Ryan Urbanowicz. Introducing Rule-Based Machine Learning: Capturing Complexity. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, GECCO '16 Companion, page 305–332, New York, NY, USA, 2016. Association for Computing Machinery.

41. F. Serendynski, P. Cichosz, and G.P. Klebus. Learning classifier systems in multi-agent environments. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 287–292, 1995.

42. Andrea Bonarini and Vito Trianni. Learning fuzzy classifier systems for multi-agent coordination. *Information Sciences*, 136(1-4):215–239, 2001.

43. Gabriel Robert and Agnès Guillot. A motivational architecture of action selection for non-player characters in dynamic environments. *International Journal of Intelligent Games and Simulation*, 4(1):5–16, 2006.

44. Satvik Jain, Siddharth Verma, Swaraj Kumar, and Swati Aggarwal. An Evolutionary Learning Approach to Play Othello Using XCS. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2018.

45. Stefan Rudolph, Sebastian von Mammen, Johannes Jungbluth, and Jörg Hähner. Design and Evaluation of an Extended Learning Classifier-Based StarCraft Micro AI. pages 669–681, 03 2016.

46. M. Gallagher and A. Ryan. Learning to play Pac-Man: an evolutionary, rule-based approach. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 4, pages 2462–2469 Vol.4, 2003.

47. L.M. Hercog and T.C. Fogarty. Co-evolutionary classifier systems for multi-agent simulation. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1798–1803 vol.2, 2002.

48. Luis Miramontes Hercog. Better manufacturing process organization using multi-agent self-organization and co-evolutionary classifier systems: The multibar problem. *Applied Soft Computing*, 13(3):1407–1418, 2013. Hybrid evolutionary systems for manufacturing processes.

49. C. Castillo, M. Lurgi, and I. Martinez. Chimps: an evolutionary reinforcement learning approach for soccer agents. In *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference*

Theme - System Security and Assurance (Cat. No.03CH37483), volume 1, pages 60–65 vol.1, 2003.

50. Niels Justesen, Tobias Mahlmann, S. Risi, and J. Togelius. Playing Multiaction Adversarial Games: Online Evolutionary Planning Versus Tree Search. *IEEE Transactions on Games*, 10:281–291, 2018.

51. Mandy J. W. Tak, Marc Lanctot, and Mark H. M. Winands. Monte Carlo Tree Search variants for simultaneous move games. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8, 2014.

52. Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*, GECCO'99, page 337–344, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

53. Pier Luca Lanzi and Perrucci Alessandro. Extending the representation of classifier conditions part II: From messy coding to s-expressions. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, pages 345–352, 1999.

54. Martin Volker Butz and Stewart W. Wilson. An algorithmic description of xcs. *Soft Computing*, 6:144–153, 2002.

55. Stewart W. Wilson. Compact rulesets from xcsi. In *IWLCS*, 2001.

56. Phillip William Dixon, David W. Corne, and Martin John Oates. A Ruleset Reduction Algorithm for the XCS Learning Classifier System. In *IWLCS*, 2002.

57. Mani Abedini and Michael Kirley. An enhanced XCS rule discovery module using feature ranking. *International Journal of Machine Learning and Cybernetics*, 4, 06 2012.

Acronyms

AFRL Air Force Research Laboratory. 3

AI Artificial Intelligence. 1, 6

APCS Adaptive Pittsburgh Classifier System. iv, 19, 31, 34, 35, 36, 42, 43, 44, 49, 55, 57, 62, 63, 64, 67, 68, 72

CAS Close Air Support. 26, 27

CCEA Cooperative Co-evolutionary Evolutionary Algorithm. 11

CRA Classifier Reduction Algorithm. 64

CS-1 Classifier System One. 12, 17

DCA/OCA Defensive Counter-Air/Offensive Counter-Air. 26

FFT Fast And Frugal Tree. 7, 19, 22, 62, 72

GA Genetic Algorithm. iv, 2, 10, 11, 12, 17, 19, 33, 34, 35, 37, 38, 43, 47, 69, 70

GUI Graphical User Interface. 27

HtAPCS Heterogeneous Adaptive Pittsburgh Classifier System. 37, 43, 44, 48, 49, 50, 55, 67

HtXCS Heterogeneous eXtended Classifier System. 43, 49, 53

HtZCS Heterogeneous Zeroeth-Level Classifier System. 42, 49, 51, 52, 54

IEEE Institute of Electrical and Electronics Engineers. 3

LCS Learning Classifier System. iv, 2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 29, 31, 32, 33, 36, 38, 42, 43, 45, 46, 47, 48, 49, 50, 51, 62, 63, 64, 66, 68, 69, 70, 71, 72

MALCS Multi-Agent Learning Classifier System. iv, 13, 31, 36, 37, 42, 47, 62, 64, 68

MAS Multi-Agent System. 11, 13, 23

MDP Markov Decision Process. 6

MHiCS Modular and Hierarchical Classifier System. 22

MLCS Multi-Learning Classifier System. 13

MMORPG Massively Multiplayer Online Role-Playing Game. 22

NEAT Neuroevolution of Augmenting Topologies. 20

OEP Online Evolutionary Planning. iv, 33, 38, 42, 45, 46, 51, 68, 71

RDP Rule Discovery Probability Vector. 69, 70

SSM Surface-to-Surface Missile. 26

XCS eXtended Classifier System. iv, 17, 18, 20, 22, 23, 31, 32, 33, 36, 42, 48, 49, 53, 54, 63, 69

ZCS Zeroeth-Level Classifier System. iv, 17, 18, 22, 23, 31, 33, 35, 36, 42, 49, 51, 52, 53, 54, 62, 63, 64, 65, 66

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)* 24–03–2020	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sept 2020 — Mar 2022

4. TITLE AND SUBTITLE

Exploring Learning Classifier System Behaviors in Multi-action, Turn-based Wargames

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Garth J.S. Terlizzi III, 2d Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-22-M-066

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISB
26 Electronic Parkway
Rome, NY 13441
POC: Dr. Brayden Hollis
COMM: 315-330-2331
Email: brayden.hollis.1@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT State of the art game-playing Artificial Intelligence research focuses heavily on non-symbolic learning methods. These methods offer little explainable insight into their decision-making processes. Learning Classifier Systems (LCSs) provide an alternative. LCSs use rule-based learning, guided by a Genetic Algorithm (GA), to produce a human-readable rule-set. This thesis explores LCS usefulness in game-playing agents for multi-agent wargames. Several Multi-Agent Learning Classifier System (MALCS) variants are implemented in the wargame Stratagem MIST: a Zeroeth-Level Classifier System (ZCS), an eXtended Classifier System (XCS), and an Adaptive Pittsburgh Classifier System (APCS). These algorithms were tested against baseline agents as well as the Online Evolutionary Planning (OEP) algorithm. In a round-robin comparison among the agents, all LCS agents outperformed the baselines and OEP. APCS is the most effective game-playing agent while producing the most explainable output. ZCS and XCS outperformed the baselines and produced interpretable rule-sets. The results highlight the ability for symbolic methods to learn a complex wargame, outperform non-symbolic competitors, and provide replicable instructions.

15. SUBJECT TERMS

Learning Classifier Systems, Reinforcement Learning, Explainable Artificial Intelligence, Wargaming

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Lt Col David W. King, AFIT/ENG
U	U	U	UU	102	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-3636; david.king@afit.edu