University of Stuttgart
Institute of Industrial Automation and
Software Engineering

Analysis of Adversarial Multi-Agent Game Outcomes of Dynamic Fault Tree Games

Forschungsarbeit 3723

Submitted at the University of Stuttgart by

Ananthu Ramesh S

M.Sc. Electrical Engineering

Examiner: Jun.-Prof. Dr.-Ing. Andrey Morozov

Supervisor: Joachim Grimstad, M.Sc. Ing

17th December 2024

Abstract

Modern engineering systems demand complexity and while dealing with complexity, ensuring system reliability and safety is a critical goal in Model Based Systems Engineering, particularly in safety-critical environments. This study explores the use of Adversarial Multi-Agent Reinforcement Learning to analyze and improve system reliability with a prototype system modeled using a Dynamic Fault Tree (DFT). A zero-sum game is implemented with two adversarial agents, the fault injecting Red Agent and the fault repairing Blue Agent, trained using Double Deep Q-Networks within a Multi-Agent Reinforcement Learning environment. The red agent strategically injects faults into system components located at the base level known as Basic Events, while the blue agent aims to repair failures and maintain system operability.

The adversarial agents interact in a turn-based sequence, with rewards structured to incentivize system failure for the red agent or prolonged operation for blue agent. The overall experiment is conducted in PettingZoo multi-agent library. The trained models are put for testing by playing several sets of games, where the outcomes of these games are analyzed resulting in a basic decision to improve the system reliability. Event Usage, Improvement Potential and Reliability analysis are conducted using the red agent's policy to identify the critical events. The analysis aims to highlight frequently targeted events and event sequences, including combinations of basic events, which significantly influence fault propagation and Top Event failure.

The study further evaluates the critical events and introduces modifications to the initial DFT model structure to see how system performance can be enhanced through targeted interventions. The agents are retrained over the new DFT and the results are analyzed to understand the importance of improvements in system fault tolerance. Redundancy and parameter adjustments have significant impact over fault tolerance of a system. The results confirm the proposed method significantly improves system reliability by making it more tolerant to adversarial faults. Moreover, the game win percentage shows improvement over the modified DFT which implies that the Adversarial Multi-Agent Reinforcement Learning framework can be implemented to identify critical components in a system. And, improving the identified critical events enhances system reliability. Ultimately, this work underscores the potential of combining reinforcement learning with DFT modeling for more effective system risk management and provides a foundation for future developments in system reliability engineering.

die Zusammenfassung

Moderne Ingenieursysteme erfordern Komplexität, und im Umgang mit dieser Komplexität ist die Gewährleistung von Systemzuverlässigkeit und -sicherheit ein kritisches Ziel im modellbasierten Systems Engineering, insbesondere in sicherheitskritischen Umgebungen. Diese Studie untersucht den Einsatz von adversarialem Multi-Agenten-Reinforcement-Learning, um die Systemzuverlässigkeit zu analysieren und zu verbessern, wobei ein Prototypensystem mit einem Dynamischen Fehlerbaum (DFT) modelliert wird. Ein Nullsummenspiel wird mit zwei adversarialen Agenten implementiert: dem fehlerinjizierenden roten Agenten und dem fehlerreparierenden blauen Agenten, die mithilfe von Double Deep Q-Networks in einer Multi-Agenten-Reinforcement-Learning-Umgebung trainiert werden. Der rote Agent injiziert strategisch Fehler in Systemkomponenten auf der Basisebene, die als Basisereignisse bekannt sind, während der blaue Agent darauf abzielt, Ausfälle zu reparieren und die Systemfunktionsfähigkeit aufrechtzuerhalten.

Die adversarialen Agenten interagieren in einer rundenbasierten Abfolge, wobei Belohnungen so strukturiert sind, dass sie entweder Systemausfälle für den roten Agenten oder einen verlängerten Betrieb für den blauen Agenten incentivieren. Das gesamte Experiment wird in der PettingZoo-Multi-Agenten-Bibliothek durchgeführt. Die trainierten Modelle werden getestet, indem mehrere Spielsätze durchgeführt werden, deren Ergebnisse analysiert werden, um grundlegende Entscheidungen zur Verbesserung der Systemzuverlässigkeit zu treffen. Nutzungsanalyse von Ereignissen, Verbesserungspotenziale und Zuverlässigkeitsanalysen werden mithilfe der Politik des roten Agenten durchgeführt, um die kritischen Ereignisse zu identifizieren. Die Analyse zielt darauf ab, häufig angegriffene Ereignisse und Ereignisfolgen, einschließlich Kombinationen von Basisereignissen, hervorzuheben, die die Fehlerausbreitung und das Auftreten des Top-Ereignisses signifikant beeinflussen.

Die Studie bewertet weiterhin die kritischen Ereignisse und führt Modifikationen an der ursprünglichen DFT-Modellstruktur ein, um zu untersuchen, wie die Systemleistung durch gezielte Interventionen verbessert werden kann. Die Agenten werden über den neuen DFT erneut trainiert, und die Ergebnisse werden analysiert, um die Bedeutung von Verbesserungen in der Systemfehlertoleranz zu verstehen. Redundanz und Parameteranpassungen haben signifikante Auswirkungen auf die Fehlertoleranz eines Systems. Die Ergebnisse bestätigen, dass die vorgeschlagene Methode die Systemzuverlässigkeit signifikant verbessert, indem sie das System widerstandsfähiger gegenüber adversarialen Fehlern macht. Darüber hinaus zeigt der prozentuale Gewinn der Spiele über den modifizierten DFT eine Verbesserung, was darauf hindeutet, dass der adversariale Multi-Agenten-Reinforcement-Learning-Ansatz implementiert werden kann, um kritische Komponenten in einem System zu identifizieren. Die Verbesserung der identifizierten kritischen Ereignisse steigert die Systemzuverlässigkeit. Letztlich unterstreicht diese Arbeit das Potenzial, Reinforcement-Learning mit DFT-Modellierung zu kombinieren, um ein effektiveres Risikomanagement von Systemen zu erreichen, und bietet eine Grundlage für zukünftige Entwicklungen im Bereich der Systemzuverlässigkeitstechnik.

Preface

This research work delves into the analysis of Multi-Agent game outcomes of DFT games. The motivation behind this work stemmed from the growing need to understand and improve system reliability, especially in contexts where fault propagation and repair strategies are key components of a system's operational success. This report represents the culmination of the past few months of dedication. I hope that it contributes to the ongoing research and inspire further research in this area.

I would like to express my deepest gratitude to all those who have supported and guided me throughout the course of this research. First and foremost, I would like to extend my heartfelt thanks to my supervisor, Joachim Grimstad, M.Sc.Ing, whose invaluable guidance and support has helped me to complete this project successfully. I appreciate the weekly discussions we had that shaped the overall trajectory of this project. I thank Jun.-Prof. Dr.-Ing. Andrey Morozov, for giving me this opportunity.

I also wish to thank the members of the IAS at the University of Stuttgart for providing a conducive academic environment and offering the necessary resources to carry out this project. I am deeply appreciative of the support from my peers and colleagues, whose thoughtful discussions and suggestions helped to improve the quality of my work.

Furthermore, I would like to acknowledge the support of my family and friends, who have always believed in me and encouraged me to overcome the challenges.

Finally, I extend my sincere thanks to all the authors and researchers whose work and contributions have inspired and informed this study. Without their prior research, this project would not have been possible.

Thank you all for your contributions, kindness, and support.

Table of Contents

List of Figures

List of Tables

Abbreviations

AEC Agent Environment Cycle.

AI Artificial Intelligence.

CI Criticality Importance.

CIP Credible Improvement Potential.

CNN Convolutional Neural Network.

CTMC Continuous Time Markov Chain.

D-DDQN Dueling-DDQN.

Dec-POMDP Decentralized Partially Observable MDP.

DFT Dynamic Fault Tree.

DNN Deep Neural Network.

DQN Deep Q-Network.

EGT Evolutionary Game Theory.

ESS Evolutionary Stable Strategy.

FDEP Functional Dependency.

FTA Fault Tree Analysis.

GPU Graphics Processing Unit.

Leaky ReLU Leaky Rectified Linear Unit.

MAE Mean Absolute Error.

MAL Multi-Agent Learning.

MAS Multi-Agent System.

MBSE Model Based Systems Engineering.

MDD Multiple-valued Decision-Diagram.

MDP Markov Decision Process.

ML Machine Learning.

MSE Mean Squared Error.

MTTR Mean-Time-To-Repair.

NE Nash Equilibrium.

OOPS Object-Oriented Programming System.

PAND Priority AND.

PDEP Probabilistic Dependency.

PO Pareto Optimality.

POR Priority OR.

PPO Proximal Policy Optimization.

RAMS Reliability, Availability, Maintainability and Safety.

RAW Reliability Achievement Worth.

RD Replicator Dynamics.

RL Reinforcement Learning.

RRW Reliability Reduction Worth.

SEQ Sequence Enforce.

TD Temporal Difference.

1 Introduction

Complexity is a part of increasing integration of features, components, and preferences within systems and industry. From small-scale systems to modern, high-tech solutions, there is a growing need for well-structured architecture due to their inherent complexity. In complex systems, there are always chances for lack of design quality, limited understanding about system behavior, and poor decision-making which often leads to flawed architectures and sub-optimal system performance. Modern day systems cannot afford flaws in it and also faulty functioning for about few instances can get converted to huge loss. Model Based Systems Engineering (MBSE) has turned out to be an evidently feasible and consistent approach for modeling complex systems. Reliability and fault tolerance of components as well as the entire system is of paramount importance to avoid catastrophic failures and enhance system performance.

Dynamic Fault Tree (DFT) has emerged as a powerful modeling tool for analyzing system reliability. DFTs provide a powerful method for analyzing the reliability of complex systems by incorporating temporal dependencies and dynamic behaviors. By understanding the different types of gates, modeling temporal dependencies, and interpreting the results of a DFT analysis, valuable insights about system reliability can be studied and potential failure modes can be identified. This knowledge is essential and can be used for designing more reliable systems and implementing effective reliability improvement measures. However, even traditional DFT analysis alone often struggles to proactively account for identifying and prioritizing vulnerabilities in complex systems that impact system fault tolerance. While, failure simulation and testing is a good choice for system reliability analysis, more efficient and reliable methods are still under research.

Artificial Intelligence (AI) is making things better and faster in almost all the sectors of engineering advancements. Through deeper research in the field of different Machine Learning (ML) algorithms, advanced models are being developed that can learn from data, identify patterns ad make intelligent decisions based on its learning, depending on the tasks considered. Reinforcement Learning (RL) is a sub-field of ML that rules-out the dependence on data driven approaches. In RL, unlike supervised or unsupervised learning approaches, the model learns from its interaction with the environment aiming for a maximized cumulative reward over time. Here, the model is an agent-environment interaction where an agent's decision-making process is formulated as Markov Decision Process (MDP) and the multi-agent extension of an MDP is a Markov Game. Multi-Agent Reinforcement Learning (MARL) consist of multiple agents interacting in the same environment designed in a cooperative, competitive or a combination of both, according to the task.

This project work aims to develop an evaluation concept to analyze systems using Adversarial Multi-Agent Reinforcement Learning (Adv-MARL) techniques. To analyze the system and identify system vulnerabilities, adversarial agents are being deployed, where the agents take strategic actions to maximize their cumulative reward and win the game through trial-and-error mechanism. Over time, the agents learn from their actions and choose the best actions that yield maximum reward. A system can be modeled using a DFT where the components of the system are connected through static and dynamic gates with each connection establishing a correlated relationship with other components. The agents, represented using Convolutional Neural Networks (CNNs), are trained by Deep Reinforcement Learning techniques such as Double Deep Q-Networks (DDQN). They simulate adversarial conditions to evaluate the fault tolerance of the system. Through strategic decision-making, the adversarial agent (*red_agent*) propagate fault through the DFT structure by injecting fault into the base level events and the good agent (*blue_agent*) repair faults to maximize their respective objectives, thereby exposing critical weak points in the system.

The focus remains in the analysis and improvement of system reliability by identifying critical events, understanding their significance through Event Usage and Improvement Potential (IP) analysis. The analysis is followed by introducing modifications to the DFT structure to see how system performance can be enhanced through targeted interventions. In addition to this, the impact of redundancy and parameter adjustments on fault tolerance will also be studied.

By leveraging the synergy between machine learning techniques and DFT modeling, this study presents a robust framework for proactive risk management in critical systems, laying the groundwork for further advancements in reliability engineering and fault tolerance analysis.

2 Literature Review

2.1 Model-Based Systems Engineering

Systems keep growing in size and complexity, for which the document-based approach has proven itself to be a very incompetent and outdated method of system development (Madni and Sievers, 2018). The complexity of systems has made document-centric approaches difficult to manage due to the risk of incompleteness, missing critical information, incorrect implementation, and inconsistency. The increasing scale and complexity of systems prompted the reevaluation of traditional systems engineering approaches and led to the emergence of MBSE. MBSE is a design-process-agnostic approach that places models at the center of system development. The model being the central repository enables the interconnection of model elements that ensure consistency, traceability, error identification, and the seamless propagation of changes in design.

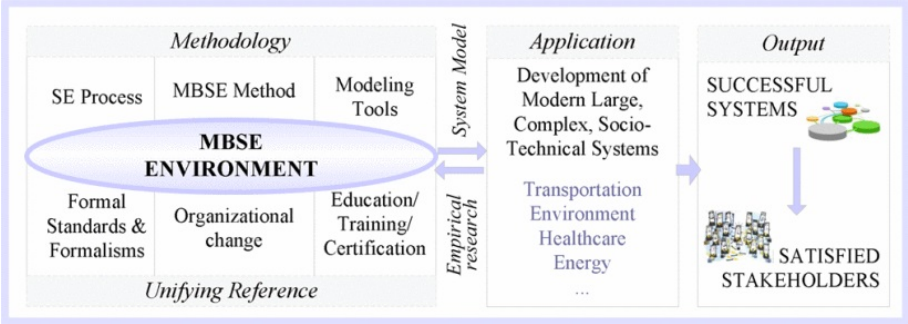


Figure 1: Integrated MBSE environment (Ramos, Ferreira and Barcelo, 2012)

Fault injection, analysis and correction in MBSE plays a crucial role in ensuring the reliability and safety of complex systems (Moradi, Acker and Denil, 2023). The purpose of this is to identify and mitigate the potential faults that are critical and could lead to system failures. Traditional fault injection methods can cause errors and increase the time and cost of safety validation as they are often dependent on human-driven activities. As a solution to this, advancements in technology particularly in the field of machine learning, the integration of reinforcement learning with high-level domain knowledge is researched a lot to enhance fault identification. With the utilization of dynamic fault models and leveraging system architecture and learned data, this innovation allows for more efficient exploration of the fault occurrences. It enables engineers to improve the safety assessment process by conducting fault injection experiments with a much improved accuracy and effectiveness. This not only helps the identification of critical faults in a complex system in MBSE but also supports the development of more reliable systems capable of operating in complex environments.

2.2 Fault Tree

2.2.1 Fault Tree Analysis

A critical step in the design of today's computer and communication systems is the reliability engineering. Failures can be fatal for safety-critical systems, such as nuclear power plants and airplanes, but they are also typically very expensive for other applications, like online ticket vending machines or any small-medium-large systems. The fault tree formalism is one of the most popular formalisms for describing and evaluating system reliability. Fault trees are represented using logical block diagrams that depict the state of a system, represented by the top event, in terms of the states of its components, which are intermediate and basic events (Boudali, Crouzen and Stoelinga, 2010). In complex systems, Fault Tree Analysis (FTA) is an extensively used technique to perform Reliability, Availability, Maintainability and Safety (RAMS) analysis. Though static fault tree is sufficient to model most features in complex systems, the expressiveness and readability of

the models are achieved through the implementation of a dynamic factors into it (Liu, Wu and Kalbarczyk, 2017).

Widely used Fault Tree related terms in this Research Thesis:

Basic Events: The most basic and indivisible elements that contribute to a system failure is generally known as a basic event (Paté-Cornell, 1984). These are the fundamental building blocks in a FTA and forms the terminal nodes in the logical hierarchy that are not further analyzed or decomposed within the fault tree. Basic events are often described as Boolean variables, sometimes referred to as binary variables, which indicate whether the event occurs or not. A system's top event probability (failure) is derived as a function of the basic event probabilities. Basic events have one output and possibly one input which can only be connected to specific gate types. They have only two states, either 0 or 1. In this project, 0 represents failure of the component, while 1 represents the component is working.

Intermediate Events: The logical combination of basic events or other events in a fault tree that represent the subsystem failures or conditions forms the intermediate events (Sinnamon and Andrews, 1997). These events are present between the basic events and the top event, which are logically coupled to any static or dynamic gates as required in a fault tree.

Top Event: It represents the overall system state. When a combination of basic event failure lead to the system failure (top event occurrence), it is known as cut set of a fault tree and the smallest possible combination of failures is the minimal cut set. Removing one of the components from the minimal cut set can prevent the system failure.

Fault Propagation: Well, as the leaves of a fault tree model component failures the gates model failure propagation (Ruijters and Stoelinga, 2015). For better understanding, an example of the fault tree components and fault propagation is given in figure 2.

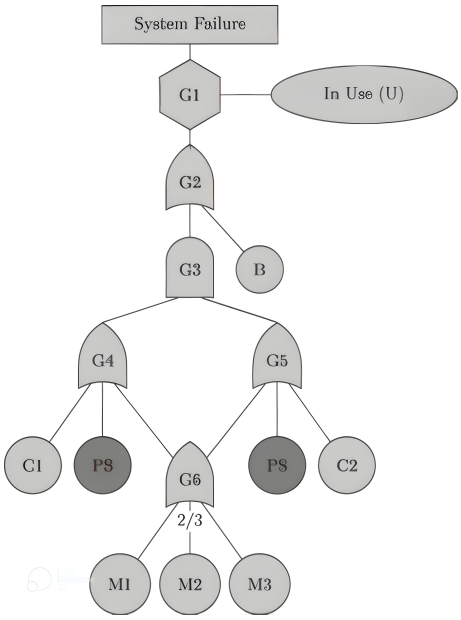


Figure 2: Example FT of a computer system with a non-redundant system bus (B), power supply (PS), redundant CPUs (C1 and C2) of which one can fail with causing problems, and redundant memory units (M1, M2, and M3) of which one is allowed to fail; failures are propagated by the gates (G1–G6). PS is somewhat darker to indicate that both leaves correspond to the same event (Ruijters and Stoelinga, 2015).

2.2.2 Dynamic Fault Tree

Dynamic Fault Tree (DFT) is an advanced technique in reliability engineering, used to analyze the reliability of complex systems. DFTs extend standard (or static) fault trees by incorporating additional gates known as dynamic gates (Boudali, Crouzen and Stoelinga, 2010). In simple terms, a DFT can be considered as a tree in which leaves form the basic events (BEs) and other elements are gates, which are widely used to represent the system failure in terms of the failure of its components. The failure is governed by probability distribution. The gates are either static gates like AND, OR, K/M voting gate or can be dynamic gates like Priority AND (PAND), SPARE, Functional Dependency gate (FDEP).

System reliability can be represented in a more comprehensive and accurate manner in DFTs, unlike the traditional fault trees, since DFTs incorporate the temporal dependencies and dynamic behaviors of system components (Liu, Wu and Kalbarczyk, 2017). DFT analysis is well focused on two major tasks. One is the qualitative analysis, which determines the (minimal) cut set of low-level failures that cause the failure at the top-level. Second is the quantitative analysis which involves computing the overall system reliability, mean time to failure of the system, etc. DFTs compute system unreliability, the probability that the system fails during mission time and other measures, such as the average time until a failure occurs (Boudali, Crouzen and Stoelinga, 2010). These dynamic gates establish accurate modeling of real-world scenarios by capturing complex fault propagation within the system and temporal dependencies. Systems having repairable components or the ones that exhibit complex interactions over time are the ones which are usually modeled using DFTs. For instance, in sequence and time critical systems there could be certain conditions or dependencies such as a component must fail before another can be affected.

Track of failure history is also very important in DFTs. The analysis of DFTs involves transforming the fault tree into mathematical models that can be evaluated using techniques like Monte Carlo simulation or state-space analysis. Identification and prioritization of potential failure modes can be achieved by interpreting the results from analysis, finally leading to a robust and fault-tolerant system design (Ruijters and Stoelinga, 2015).

Failure time is another important concept and is defined as the exact time at which a failure occurs in a system or component. Reliability and performance of a system is predicted using this failure time. They help in finding how long a system or component can operate before a failure occurs. Failure times of elementary components are assumed to be independent and often exponentially distributed. As a result, the underlying stochastic process is a Continuous Time Markov Chain (CTMC) which is analyzed numerically. Exponential distribution has the memory-less property which makes it a common choice for modeling the time between failures.

Temporal Dependencies are one of the most critical aspects of DFTs, which allows the model to capture the sequence and timing of events (Buchholz and Blume, 2022). These dependencies include Sequence Dependencies, Functional Dependencies, Repair and Maintenance. Sequence dependencies are the ones which are modeled using PAND gates and they capture the conditions that certain failures must occur in a specific order. Functional dependencies are modeled using FDEP gates. They are focused on the impact of a component failure on other components that can eventually lead to the failure of the system in its entirety. Another essential criteria for modeling the reliability of systems is the repair and maintenance actions for DFTs with repairable components.

2.2.3 Components of a DFT

Static Gates

Static gates combine component failures using logical operators (Buchholz and Blume, 2022). The most general gates are AND, OR, and K out of N gate with N inputs ($K \leq N$).

- **AND gate:** It represents a situation where output event fails if all input events fail.
- **OR gate:** In the case of OR gate, failure of any one of the input will cause failure of the output.

- **K out of N gate:** Here the output becomes 0 (indicates failure) if K out of N inputs fail and any additional failures does not change the output. With respect to this, AND gate is a N out of N gate and the OR gate is a 1 out of N gate.

AND and OR gates defines the maximum and minimum of the failure time distributions at the input, respectively. Static gates have a single output and $N \geq 2$ inputs.

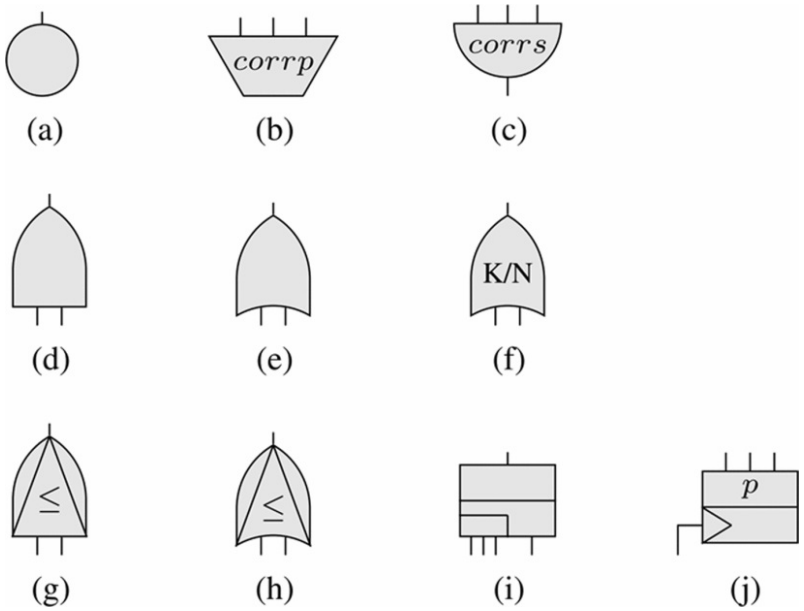


Figure 3: DFT gate types: (a) basic event, (b) parallel correlator, (c) sequential correlator, (d) AND gate, (e) OR gate, (f) K out of N gate, (g) PAND gate, (h) POR gate, (i) Spare gate, and (j) PDEP gate (Buchholz and Blume, 2022).

Dynamic Gates

The gates become dynamic when the gates have some conditions or dependencies for and when the failure occurs. The different dynamic gates are PAND, Priority OR (POR), Spare gate, Probabilistic Dependency (PDEP) and FDEP gates.

- **PAND gate:** It only fails if the inputs fail from left to right in order. Thus, the gate will never fail if one component fails out of order, and it is used to model sequence-dependent failures.
- **Priority OR gate:** If the leftmost input fails first, then only the POR gate will fail. PAND and POR gates have a single output and $N \geq 2$ inputs.
- **Spare gate:** The spare gate has one output, one primary input, and any number of secondary inputs ordered from left to right. The leftmost available spare substitutes the primary input in case of failure of the primary gate. If that component also fails, then it is substituted by the next available spare. This process is denoted as claiming. It models redundancy and standby components. A component can be claimed by only one spare gate and if no component is available to be claimed, the spare gate fails. Earlier, only basic events were allowed as inputs to spare gates, but nowadays disjoint sub-trees are also used as inputs. Components that are used as a spare, which are basically in the dormant mode, may sometimes fail. In this case, the corresponding component is a warm spare. On the other hand, a cold spare is a component which can not fail if it is not used.
- **PDEP and FDEP gate:** The PDEP gate has a single primary input and $N \geq 1$ outputs, with a single parameter $p \in [0, 1]$. Each of the outputs of the PDEP gate fails immediately after the primary input fails with a probability p. The gate has no effect when $p = 0$.

As p becomes 1, the gate is denoted as an FDEP gate. This implies that whenever the primary input fails, all outputs also fail immediately. In DFTs, PDEP and FDEP gates can be substituted using the sequential correlator, since it is an extension of the PDEP gate. Consider if the input of the sequential correlator fails at time t', the conditional cumulative distribution function is used to determine the output of the remaining time to failure of basic events connected to the output, represented by $F(t - t'|t')$. The time to failure of the components connected to the output is determined by failure time t'. If $F(0|t') = 1$, then the sequential correlator is equal to an FDEP gate. Dependencies are symmetric in the case of a parallel correlator, while on the other hand, a sequential correlator is involved in the modeling of situations where the event at the output depends on the event at the input but not vice versa.

2.2.4 A simple DFT example

The system shown in Figure 4 consists of three components, each described by two elementary components connected via an OR gate. Two of the components are operational components, while the third component acts as a warm spare for both of the others. Two versions of the system can be considered. As shown in the figure, one of the versions has one functioning component that is required for the system to be operational. In the second version, the top-level event becomes an OR gate such that two operational components are necessary for the system to be operational. The elementary components of the first type are c_{11}, c_{21}, and c_{s1}, and the elementary components of the second type are c_{12}, c_{22}, and c_{s2}, with correlated failure times that are modeled using parallel correlators.

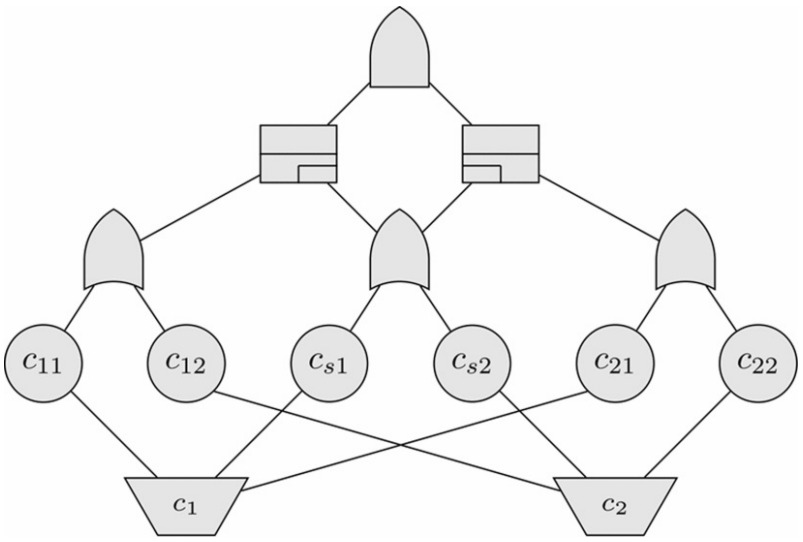


Figure 4: Simple example DFT with correlated basic events (Buchholz and Blume, 2022).

In simple terms, the system components are c_1, c_2, and a warm spare. Each main component has two elementary components. The structure of the DFT consists of a gate at the top of the tree that determines the system's operational state. In the given diagram, this top gate is an AND gate, which states that one functioning component is required for the system to be operational. The second version is with an OR gate, indicating two operational components are necessary for a working system. Each of the middle OR gates represents a main component, either c_1 or c_2, and the warm spare. These gates are the ones that connect the two elementary components that eventually result in each of the main components. The parallel correlators between c_{11}, c_{21}, c_{s1} and between c_{12}, c_{22}, c_{s2} indicate that the failure times of these components are correlated.

In the second version, where an OR gate is at the top-level, for the system to be considered functional, it would require two operational components instead of just one. This structure is an efficient technique for modeling the reliability of complex systems by considering both the redund-

ancies—one provided by the warm spare and the other by correlated failure times of components. A DFT has exactly one top-level gate, which is not a PDEP gate or a correlator. It has an output that describes the system state and is not connected to any input. If it becomes 0, then it indicates that the system has failed.

2.2.5 Modularization Approaches

Modularization techniques are widely used in DFTs (Basgöze et al., 2022). It's a "divide-and-conquer" based approach which splits the DFT into modules, i.e., independent sub-trees. Now, individual modules can be analyzed independently and the results are combined at the end. The sub-modules which has dynamic elements are analyzed by translating them into a Markov model. All possible BE failures in the DFTs are explored to create a state space of the Markov model. Each transition represents the failure of BE and the subsequent state represents the status of the DFT after the failure. The transition rate is determined by the failure rate of the BE.

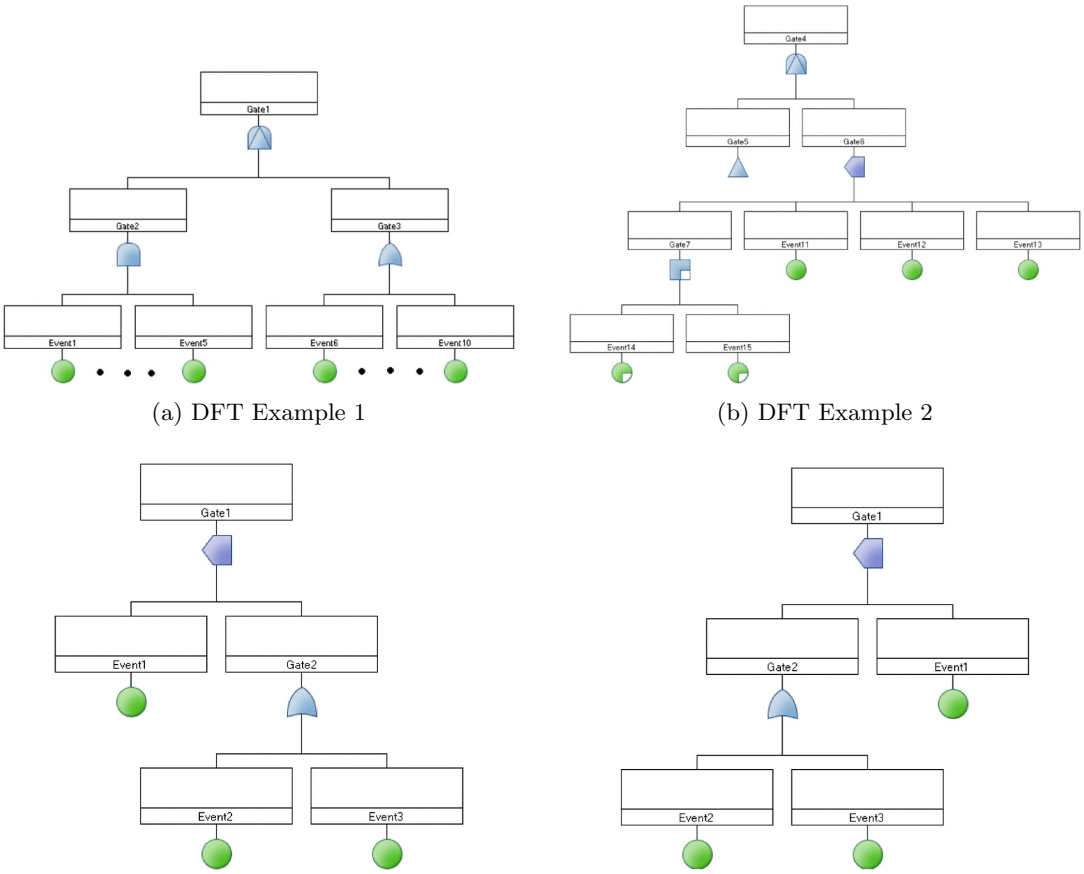
The conversion of a DFT into a Continuous-Time Markov Chain is a very common approach for solving DFTs. Calculation of system reliability and failure probabilities are achieved by steps that involves representing the system states and transitions over time. Modularization techniques are used for the efficient reduction of both dynamic and static fault trees by decomposing them into smaller and independent modules (Yevkin, 2011). Doing so would reduce the computational complexity by individually analyzing each independent sub-tree. Consider a case where an OR gate is present inside a DFT module comprising multiple independent basic events with constant failure rates, they can be combined into a single equivalent event with a cumulative failure rate. Basic events can be used in terms of internal modules, if this can lead to a DFT with only static or PAND gates, which simplifies the conversion to semi-Markov Chains. Reduction of irrelevant states in DFTs, if it has only PAND and static gates, can be achieved during Markov Chain generation. Thus, streamlining the analysis. Simplification of a dynamic module with the top event as a sequence enforcing (SEQ) gate can be done by reordering components if the first input event to the gate is an independent event. The complexity can be further reduced by substituting the first independent input event of a spare gate by a basic event, facilitating further reduction in complexity.

2.2.6 Demonstration of Modularization using Examples

Example 5a demonstrates the application of modularization in a fault tree having a PAND gate at the top with AND and OR gates as inputs. Here,

- The first step simplifies the OR gate, which consists of multiple independent basic events with constant failure rates. By following the modularization approach, the OR gate is replaced by a single equivalent event with a constant failure rate, calculated by summing the failure rates of the individual events. This substitution streamlines the fault tree, combining several events into one without changing the system's overall behavior.
- The next step focuses on the AND gate, which is static and independent from the rest of the fault tree. Since there is a lack of dynamic behavior in it, it can be analyzed using standard static fault tree methods. The resulting failure probabilities at various time points are then approximated using a Padé function, which helps in converting the probability data into a failure rate function for further analysis.
- After handling the OR and AND gates, the PAND gate is left with two basic events, each with time-dependent failure rates. This setup is transformed into a continuous-time semi-Markov chain. Using the Runge-Kutta-Fehlberg method, the chain is solved with just three states, reducing computational effort. The state transitions capture the dynamic behavior dictated by the PAND gate's logic.

This modularization approach significantly reduces the number of states of Markov Chain required for analysis by effectively breaking down a complex DFT into manageable components.



(a) DFT Example 1

(b) DFT Example 2

(c) DFT Example 3 with SEQ gate

(d) Example 4 Equivalent DFT

Figure 5: DFT Examples to Demonstrate Modularization (Yevkin, 2011)

In example 5b, a more complex DFT is simplified with nested dynamic gates and static gates, reducing computational requirements from 19,456 states to a manageable sequence of simpler Markov Chains. Example 5c and 5d shows the use of Padé functions to approximate failure rates in a DFT with SEQ and OR gates. The traditional Markov Chain analysis becomes very complicated when analyzing fault trees with complex failure rate distributions especially when the BEs have non-constant failure rates. In this example the fault trees are restructured and an equivalent fault tree with a static sub-tree is reformulated which helps in calculating the system unreliability even more effectively. By using this methodology, the accuracy and efficiency of the method can be improved.

Limitations of Modularization

To reduce the state space and to improve the computational efficiency of the Markov model underlying a DFT model, modularization was introduced (Mo, 2014). Each statistically independent sub-tree of a fault tree is detected and replaced by a single event whose probability of occurrence represents the probability of occurrence of the sub-tree. In complex real-life DFT models, the repeated failure events can cause the traditional modularization-based DFT analysis processes to generate large dynamic sub-trees, which can lead to a state explosion problem.

Multiple-valued Decision Diagram: To solve the above stated limitation and for computing the reliability of large dynamic sub-trees an efficient, multiple-valued decision-diagram (MDD)-based DFT analysis approach was proposed by Yuchang Mo. In traditional modularization methods the whole dynamic sub-tree must be solved using state-space methods, the proposed approach limits the use of complex calculations to only the parts of the system where dynamic failure behaviors are present within the dynamic sub-tree. Here, a dynamic gate is represented in such a way that it can exist in multiple states rather than just a simple failed or operational state. With the use of this technique, a single compact MDD can be used for modeling the failure behavior. This avoids

the state-space explosion normally incurred in conventional methods.

2.3 Reliability Analysis and Importance Measures

Reliability analysis is the most crucial part in identifying vulnerabilities within engineering systems (Amrutkar and Kamalja, 2017). It helps to ensure consistent performance and prevent unexpected failures. Understanding the importance of a component focusing on determining how critical each individual components are and how they contribute to the overall reliability of a system is the motive of this analysis. Prioritizing improvements and optimizing maintenance strategies are very important for maintaining a reliable system. In real-world applications, the component with lower reliability is usually more critical, as it poses a greater risk to system performance. The concept of component importance was first introduced by Birnbaum in 1969. Since then, researchers have developed various methods to measure and rank the significance of different components within a system. These importance measures assign numerical values to components, enabling the identification of those most crucial to reliability enhancement or those most likely to contribute to system failure. Providing a clear hierarchy of component criticality makes these measures an essential tool for reliability engineers to enhance system robustness by mitigating potential risks.

Over the years, numerous studies have expanded on the initial framework proposed by Birnbaum, introducing refined measures tailored to different types of systems and failure behaviors. These measures apply to both traditional and modern systems, integrating advanced techniques such as sensitivity analysis, probabilistic risk assessment, and mathematical programming. The results of these studies draw attention to the growing importance of evaluating component importance not only in static systems but also in complex, dynamic environments. Birnbaum's measure focuses on the potential impact of making a component perfectly reliable, while other methods assess the effects of component interactions, sequence of failures, or time-dependent behaviors. Component importance analysis is a cornerstone of reliability engineering, offering valuable insights into system weaknesses and guiding efforts toward targeted reliability improvements. By exploring the development and application of different importance measures, we can better understand how to enhance system resilience and ensure optimal performance across a wide range of industries.

Some of the importance measures are discussed in the following sections and will be used in the evaluation section of this project.

2.3.1 Birnbaum's measure

The Birnbaum's measure of importance, denoted as I^B, evaluates the impact of a component on overall system reliability by calculating the maximum reduction in system reliability when that component transitions from perfect functionality to complete failure. It shows how critical a component is to the system's performance. The mathematical representation of Birnbaum's measure is given in equation 1 (Espiritu, Coit and Prakash, 2007):

$$I_i^B(t) = \frac{\partial R_s(t)}{\partial R_i(t)} = R_s(t; R_i(t) = 1) - R_s(t; R_i(t) = 0) \tag{1}$$

where,

- $I_i^B(t)$ is the Birnbaum importance of component i.
- $R_s(t)$ is the system reliability at time t.
- $R_i(t)$ is the reliability of component i at time t.
- $R_s(t; R_i(t) = 1)$ is the system reliability at time t given that component i is perfectly working.
- $R_s(t; R_i(t) = 0)$ is the system reliability at time t given that component i is failed.

Birnbaum's importance measure does not take into account the actual reliability $R_i(t)$, or unreliability $F_i(t)$, of a component, which can be considered as its weakness or limitation. As a result, even if the reliability levels differ significantly for two components, they can still have similar Birnbaum importance values.

2.3.2 Criticality Importance

Criticality Importance (CI) is an extension of the Birnbaum metric which solves the limitation of Birnbaum's measure by integrating the component unreliability $F_i(t)$ as shown in equation 2 (Espiritu, Coit and Prakash, 2007). By doing so, the component with less reliability is given more importance and priority as it is more critical.

$$I_i^{CR}(t) = I_i^B(t) \left(\frac{F_i(t)}{F_s(t)} \right) = [R_s(t; R_i(t) = 1) - R_s(t; R_i(t) = 0)] \left(\frac{F_i(t)}{F_s(t)} \right) \tag{2}$$

where $F_s(t)$ represents the system unreliability at time t.

2.3.3 Reliability Achievement Worth and Reliability Reduction Worth

These are defined by the reliability and not by the risk. The Reliability Achievement Worth (RAW) gives a measure of how much a component impacts the system reliability when a component is perfectly reliable. While, Reliability Reduction Worth (RRW) considers the impact of a component on the system when it is perfectly unreliable, that is, the component always fails. RAW and RRW are mathematically represented as equation 3 (Espiritu, Coit and Prakash, 2007) and equation 4 (Espiritu, Coit and Prakash, 2007) respectively.

$$RAW_i = \frac{R_s(t; R_i(t) = 1)}{R_s(t)} \tag{3}$$

$$RRW_i = \frac{R_s(t)}{R_s(t; R_i(t) = 0)} \tag{4}$$

2.3.4 Improvement Potential

Improvement Potential (IP) as stated from the name itself defines the potential of a component to improve the system reliability. Consider there are n number of components in a fault tree. These components can be arranged into a list where the component with the highest value of IP has the highest contribution to the reliability of the system. By improving this component improves the system reliability the most. With the help of such an analysis it is very convenient to identify which all components needs to be improved in a system and which all are the ones that are not that important.

IP is defined as the difference between system reliability when a component i is perfectly working, ie., the component i with a reliability 1 (or failure rate of 0), and the system reliability with the actual failure rate of the component. IP can be calculated using the formula depicted in equation 5 (Catelani, Ciani and Venzi, 2016).

$$I_i^{IP}(t) = R_s[t; R_i(t) = 1] - R_s(t) \tag{5}$$

where $I_i^{IP}(t)$ is the improvement potential of the component i.

A reliability of 1 is never possible in reality, it is an assumed value for calculation. In $R_s(t)$, the actual failure rate and the corresponding reliability of the event or component is considered.

The assumed value of reliability 1 for a particular component under consideration is said to be a limitation of IP since none of the physical components can be one hundred percent reliable. For this reason, a new component measure was introduced known as Credible Improvement Potential (CIP), to mitigate the limitation. It is defined based on the equation 6 (Catelani, Ciani and Venzi, 2016):

$$I_i^{CIP}(t) = R_s[t; R_i(t) = R_i^+(t)] - R_s(t) = \Delta R_s(t) \tag{6}$$

where $I_i^{CIP}(t)$ is the Credible Improvement Potential of component i at time t and $R_i^+(t)$ represents the new improved reliability of component i at time t.

2.4 Game Theory

Game theory is a mathematical framework which have been widely used to model strategic interactions and their likely outcomes among independent agents (Rizk, Awad and Tunstel, 2018). It has been used across various fields from economics, psychology to artificial intelligence. Here the outcome for each agent is dependent not only on their own decisions but also on the decisions made by other acting agents. In general, it is a multi-agent decision-making scenario, where the scenario can be described as a game. The decision-makers are the players and their moves are the actions (Merrick et al., 2016). These actions are accompanied with a payoff which is completely dependent on their move and the moves of other players in the game.

To put forth the best strategies for players, the equilibrium points are identified by analyzing the games thoroughly. It is also very important to note that the modeling of the players needs to go a step further into the details like their intent, objectives, strategies and also their motives, to better understand the scenario and to improve the decision-making in that scenario. An advantage of multi-agent decision-making strategy is that, large number of potential threats in a game can be examined. Also, future attacks can be predicted and controlled by suggesting probable actions with certain predicted outcomes (Roy et al., 2010).

2.4.1 Games

In a game, the decisions made by some players affect the payoff of other players. A game can be best described by the decision-makers, the actions they take, and the resulting payoffs from each outcome (Merrick et al., 2016). A game can be of several types, like static, sequential, dynamic, perfect/imperfect information, complete/incomplete information, Bayesian, stochastic, cooperative/non-cooperative and zero-sum games. Out of these, stochastic games and Bayesian games are the most common games for cooperative multi-agent systems.

Strategies: Strategies are the blueprint of a game. It outlines the actions that needs to be taken during a game and when these actions is to be taken. One common strategy is that the players would like to take an action which produces the best payoff at that moment in the game. Multiple interaction between the agents over time leads to a Nash Equilibrium when both the players do this. Nash Equilibrium is an important topic with respect to game theory and will be discussed in detail in the following sections of this literature survey.

Some of the game types are discussed below:

1. Stochastic or Markov games: They are sequential probabilistic games which extent the concept of repeated games (Rizk, Awad and Tunstel, 2018). Here, the same game, chosen from a set of normal form games, is played multiple times with the outcome of one game potentially affecting the next. Stochastic games having a finite number of players, actions and states will always have an equilibrium state known as Nash Equilibrium. It is a strategic method to maximize each player's utility knowing the strategy of other players in the game.

2. Bayesian Games: The games played with incomplete information are termed as Bayesian Games, where the uncertainties are the uncertainties in the agent's payoffs. It could also be that one player doesn't know the other players' preferences, resources, or strategies. In such

games, a Bayesian Nash Equilibrium condition occurs, where each player adopts a strategy that is the best response to their beliefs about the other players' types. These beliefs and strategies form a stable state, and none of the players will be willing to change their strategy as it is the best response to their beliefs about other players.

3. Zero-sum game: A zero-sum game is the one in which each player is directly opposed by the other player (Merrick et al., 2016). Here the gains of one player can be the losses of the other player. It's a win or loose scenario, where the payoffs of one player is the negative of the other player in the game. The total gain of one player is equal to the total loss of the opponent, making the sum of their net changes to be zero.

	a1	a2
a1	(1, -1)	(-1, 1)
a2	(-1, 1)	(1, -1)

(a)

	a1	a2
a1	(5, 5)	(0, 10)
a2	(10, 0)	**(1, 1)**

(b)

	a1	a2
a1	**(5, 5)**	(0, 0)
a2	(0, 0)	**(10, 10)**

(c)

	a1	a2
a1	**(2, 1)**	(0, 0)
a2	(0, 0)	**(1, 2)**

(d)

Table 1: Examples of 2-player, 2-action games. (a) Matching pennies, a purely competitive (zero-sum) game. (b) The prisoner's dilemma, a general sum game. (c) The coordination game, a common interest (identical payoff) game. (d) Battle of the sexes, a coordination game where agents have different preferences. Pure Nash equilibria are indicated in bold (Nowe, Vrancx and De Hauwere, 2012).

2.4.2 Decision Making

One of the integral part of all these technologies coming into a collaborative environment is to achieve good decision-making (Rizk, Awad and Tunstel, 2018). It is the goal with which these ideas are driven. Decision making, or in other words planning and control, is the part of the system which pushes the agents to achieve its goals by deciding on what actions needs to be performed. It can be episodic or sequential depending on the output it produces. If the output is a single action, then it's episodic, while the other produces an output which is a sequence of actions or policy. The algorithm for decision making are evaluated based on policy optimality, search completeness, time complexity and space complexity. An optimal policy means the one which has the highest utility. Search completeness refers to a search algorithm that is guaranteed to find the best possible solution or policy within a finite amount of time, if such a solution exists. The amount of time required to search a particular solution is represented as time complexity and the computational memory required is the space complexity.

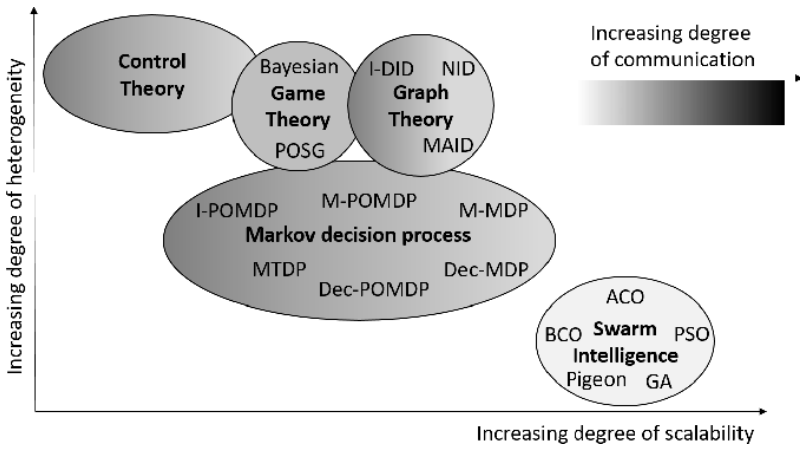


Figure 6: Comparison of decision making model frameworks (Rizk, Awad and Tunstel, 2018).

Challenges in decision making: Environment uncertainty, diversity, system scalability, real-time processing, and limited computational resources are some of the challenges that needs to be taken into account while designing a decision making algorithm.

2.4.3 Model Checking Markov Models

Model checking is a technique used to check if a given system model satisfies a certain property or specification. It involves techniques that systematically explore all possible states of a model to check if it satisfies desired properties. Markov models are used to describe the probabilistic transitions between different states of a system. In a Markov model, the future state of the system depends only on the current state and not on the sequence of events that preceded it. This property is known as the Markov property. The application of model checking techniques specifically to Markov models is known as Model checking Markov models. In the context of fault tree analysis, this kind of an approach uses model checking algorithms to analyze the behavior and properties of Markov models that represents the DFTs. In fault tree analysis while applying model checking to Markov models, the aim is to verify the reliability and performance characteristics of a system by examining the probabilistic transitions between different states captured in the Markov model.

2.4.4 Markov Decision Processes

Markov Decision Processes is a discrete time stochastic control process where at any given point the exact condition or state of the situation is observable, and the outcomes or results are dependent on the decision-makers (Rizk, Awad and Tunstel, 2018). It follows the Markov property. For a MDP, optimal and sub-optimal policies are determined to maximize the expected rewards and also helps in handling complexities in the system.

Reinforcement Learning is expressed using MDP as shown in figure 7.

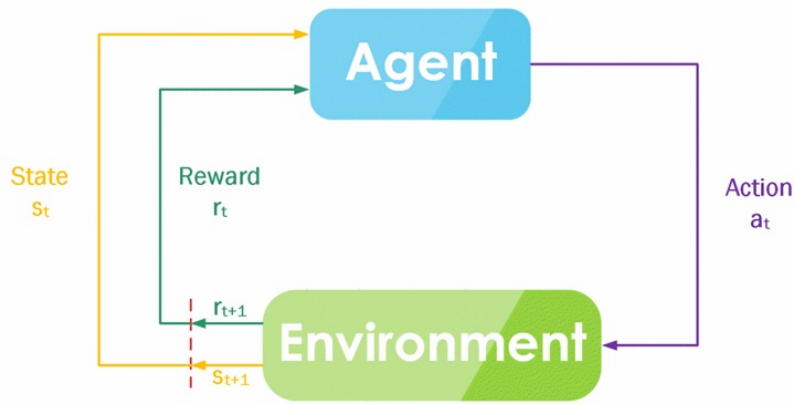


Figure 7: Markov decision process (AlMahamid and Grolinger, 2021)

In this project, the RL framework is rooted in the principles of MDPs. Each agent models the environment as a Markov process:

- **States:** state_memory and new_state_memory in the class ReplayBuffer store the current and next state transitions experienced by the agent when it interacts with the environment.
- **Actions:** The choose_action() method in the class Agent defines the agent's next move with a higher exploration rate at the beginning, and later, shifting to the exploitation of learned strategies.
- **Rewards:** The significance of an action taken by the agent in a given state is defined by the reward it gets. The best actions are rewarded more, and all the rewards are stored in the reward_memory.
- **Transition dynamics:** The environment manages state transitions implicitly as part of the simulation process through the step() method.
- **Discount factor (γ):** There should be a balance between immediate rewards and future rewards, thus encouraging the agent to optimize its performance over the long term. This balance is achieved through the discount factor.

There are several algorithms to determine state transition and reward functions in a MDP which can be very complex. As a result, approximate methods are used in which model-free RL approaches are more prominent. Distributed optimization methods are also used to solve MDP problems. Here, the MDP is decomposed into sub-problems and a distributed Newton method or linear programming algorithm is used to find the optimal policy (Rizk, Awad and Tunstel, 2018). Algorithms like Double Deep Q-Learning (DDQN), as used in this project work, eliminates the need for a predefined model and enables learning from experiences of the agent in the defined environment. MDPs are extended to multi-agent systems where two agents (red and blue) interact within a shared environment. In this case:

- A joint action space exist where the decision of one agent impacts the state transitions and rewards of the other agent. This exemplifies the complexity of multi-agent MDPs, where the dimensionality increases with the number of agents, increasing the complexity of the model.
- Complexity is managed by using a decentralized learning process with each agent maintaining its own observations, policies, and memory buffer while interacting within a partially observable environment.
- Even though this project work implementation assumes a fully observable environment, the framework could also be adapted for Decentralized Partially Observable MDPs (Dec-POMDPs), where agents make decisions based on their own limited observations if full observability is not possible. This would involve incorporating communication and information sharing methods, mechanism for partial observability and decentralized decision making.

The adoption of Deep Learning approaches are aiding to solve the limitations of MDP models of which the most predominant one is the extension of MDP from discrete to continuous space, as it is more suitable in the case of robotic multi-agent games.

2.4.5 Nash Equilibrium

A strategy profile is considered to be a Nash Equilibrium (NE) if for each player in the game, their chosen strategy is the best response to the strategies selected by the other players. This means that while playing a NE, none of the individual player in the game will be able to improve its payoffs by making a decision on their own without considering changes by the other players (Nowe, Vrancx and De Hauwere, 2012). No player can improve their payoffs by changing their strategy alone, since their current strategy is already the best response to the strategies of others. Therefore, the strategies of multiple players needs to be changed at the same time in order to escape the NE to achieve a better outcome.

Coordination games, such as in table 1(c), though it does not correspond to the global optimum, the NE corresponds to a local optimum for all players. This can be clearly seen from the game where there are two Nash equilibria, one which gives a reward of 5 for both the players and the other with a reward of 10, which is the global optimum. In the case of (b), the cooperative outcome is not a NE since both the players can betray the other and improve their payoff. The first game (a), which is a competitive game, does not have a pure strategy NE, since there exist no such pure strategy which is a best response to another pure best response. The NE for this game is that both the players should choose both the sides of the coin with equal probability. In Multi-Agent Systems (MAS), Nash Equilibrium is defined as in equation 7 (Wang et al., 2022),

$$\forall \pi_i \in \Pi_i, \forall s \in S,\, U_i(\pi_i^*, \pi_{-i}^*) \geq U_i(\pi_i, \pi_{-i}^*), \quad \forall i \in \{1, \ldots, n\} \tag{7}$$

where π^* is a joint policy, $U_i(s)$ is the expected long-term payoff of agent i in state s, and Π_i is the set of all possible strategies for agent i.

2.4.6 Pareto Efficiency

Pareto Optimality (PO) ensures that no agent in a system can improve their outcomes without sacrificing the outcomes of another agent. This is particularly relevant in multi-objective optimization problems, where agents must balance multiple conflicting objectives. In these scenarios, a solution is said Pareto optimal if no other solution can improve one agent's outcome without reducing the outcomes of others. This definition of PO is similar to the concept of global optimality in single-objective optimization, but it operates in a multi-dimensional space where multiple objectives must be considered simultaneously (Le and Ta, 2024).

Pareto stationary: x^* is Pareto stationary (equation 8) if,

$$\text{range}(JF(x^*)) \cap \mathbb{R}_{++}^n = \emptyset \tag{8}$$

where $JF(x)$ is the Jacobian matrix of $F(\cdot)$ at x and $\mathbb{R}_{++}$ is the set of positive numbers. This condition ensures that no further improvements can be made simultaneously in all objectives.

(Strong) Pareto Optimal Solution: A solution x^* is considered Pareto optimal (equation 9) if there does not exist any other solution x such that,

$$F(x) \preceq F(x^*) \quad \text{and} \quad F(x) \neq F(x^*) \tag{9}$$

No other solution can strictly improve the performance of all objectives without harming the performance of at least one.

Weak Pareto Optimal Solution: A solution x^* is a weak Pareto optimal solution (equation 10) if there does not exist any other x such that,

$$F(x) \prec F(x^*) \tag{10}$$

There is no other solution that strictly improves all objectives simultaneously.

ϵ-Pareto Optimal Solution: A solution x^* is an ϵ-Pareto optimal solution (equation 11) if there exists a Pareto optimal solution x such that,

$$F(x^*) \preceq F(x) + \epsilon \tag{11}$$

x^* is close to a Pareto optimal solution, but slight deviation from optimality is allowed within the tolerance level.

The difference between strong and weak Pareto optimality has a significant role in optimization problems. Strong Pareto optimality requires strict non-improvability and weak Pareto optimality allows for some improvements in certain objectives. Under certain conditions, weak and strong Pareto optimal solutions become equivalent, such as strong quasi-convexity and continuity of the objective functions.

2.4.7 Competitive and Cooperative Interactions in Game Theory

In game theory, games are focused on providing a systematic approach to complex systems to decide a series of Pareto-efficient strategies in cooperative games or the best strategies in non-cooperative games (Wang et al., 2022). With respect to the temporal structure and dependencies, the games are divided into static and dynamic games.

Cooperative games which is a positive sum game, work in cooperative strategies to achieve their common goal. In static cooperative games, players focus on minimizing their costs by helping each other without compromising their own payoffs. This leads to Pareto-optimal solutions in games. Dynamic cooperative games are the ones in which to achieve a common goal the players take different actions. It is a Markov game and Multi-Agent Reinforcement Learning (MARL) technique is used in modeling and analysis. The cooperative Markov games are modeled to maximize the accumulation of rewards, and also the Pareto optimal solutions are reduced to a unique optimal solution.

Non-cooperative games or competitive games are focused on maximizing individual payoffs and minimizing individual costs in order to achieve NE. In static non-cooperative games, the players' strategies are executed at the same time. Since they do not have the information about the choices of the other player, they need information to make decisions. Depending on the availability of information especially about the other players' characteristics, strategy spaces and cost functions, static non-cooperative games are further divided into complete and incomplete information games. The equilibrium of these two games corresponds to NE and Bayesian Nash Equilibrium.

2.4.8 Evolutionary Game Theory and Evolutionary Stable Strategies

Evolutionary Game Theory (EGT): How individuals within a group interact and how these interactions shape the dynamics of the moves and countermoves is the base of EGT. The rules and strategies keep evolving with respect to how other players in the group are playing. The game is a continuous process of actions and counter-actions and this back-and-forth actions results in a constantly changing game environment (Jiang et al., 2019). Over time, trial-and-error experiments and experience lead to exploring the strategy or strategies that work really well. This dynamic process of learning and adapting leads to an equilibrium.

Evolutionary Stable Strategies (ESS): A strategy is said to be an ESS (Tuyls, Hoen and Vanschoenwinkel, 2006) if it can survive and resist any evolutionary challenges from newly developing mutant strategies.

Limitations of Nash Equilibrium

In recent years, there is a shift in game theory away from classical solutions like NE towards EGT concepts like ESS and replicator equation (Tuyls and Parsons, 2007). This is because,

- NE is computationally complex and is unable to handle highly dynamic scenarios.
- Often fails to provide the most optimal or stable strategies for players.

Advantages of EGT

Compared to the normative approach of traditional game theory, EGT suffers less from these issues because its descriptive approach aligns in a much better way with the overall goals of multi-agent learning compared to the normative approach of traditional game theory.

- Unlike Game Theory, where it is assumed that the players in a game will calculate the Nash equilibria and choose to play on one of these equilibrium strategies, EGT assumes that with the repeated observations of own payoffs and that of others players are expected to adjust their strategies over time.
- This type of learning is governed by Replicator Dynamics (RD), which sets the frequency of different pure strategies based on the current mix of strategies in the population.
- Average payoff of the game, depending on the type of game, is calculated and strategies that have an above-average payoff are the one which are likely to be executed.
- RD models a process in which strategies that appear to be the most successful are adopted by the agents.

2.5 Reinforcement Learning

Understanding the fundamentals is the core idea behind any kind of learning, whether it is at the individual agent level or at a multiple agent level. In Reinforcement Learning, agent interacts with other agents and the environment, and what it learns from these interactions is the fundamental basis (Tuyls, Hoen and Vanschoenwinkel, 2006). RL involves discovering what to do so that the numerical reward signal can be maximized by learning how to map situations to actions. RL can be classified based on number of states and action types depicted in figure 8.

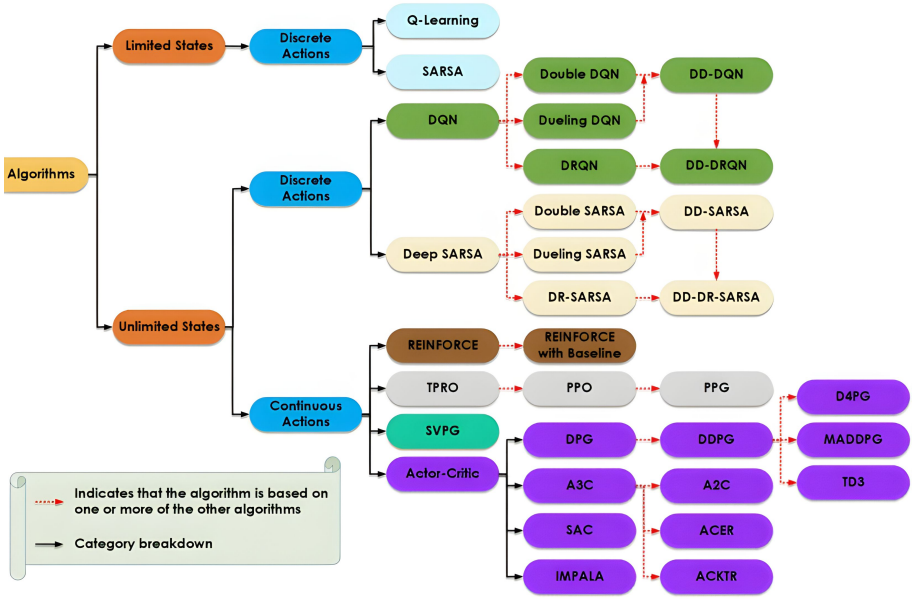


Figure 8: Classification of RL Algorithms (AlMahamid and Grolinger, 2021)

Actions taken by the agents not only affect the current reward but also the upcoming situations and the corresponding rewards as well. The two most important features of RL are the trial-and-error search and delayed reward.

An agent's (Tianxu Li et al., 2022) decision-making process is formulated as MDP, which is defined by a tuple $\langle S, A, R, T, \gamma \rangle$ where:

- S is the finite set of states
- A is the finite set of actions
- $R : S \times A \times S \to \mathbb{R}$ (rational number $\mathbb{R}$) is the reward function which determines the immediate reward that the agent receives after executing an action $a \in A$ given state s.
- $T : S \times A \times S \to [0, 1]$ is the transition function that denotes the probability of a transition from state $s \in S$ to next state $s' \in S$ after an action $a \in A$ is taken.
- $\gamma \in [0, 1]$ is a discount factor which is used to trade off the immediate rewards and the future rewards.

At a time t, the agent takes an action a_t in a state s_t that results in a system transition to $s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t)$ and receives a reward $R(s_t, a_t, s_{t+1})$ (Zhang, Yang and Başar, 2021). The main objective in MDP is to determine a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ where $\Delta(\mathcal{A})$ is the distribution of action space $\mathcal{A}$, such that $a_t \sim \pi(\cdot \mid s_t)$ and the discounted long term reward value can be maximized. This reward is represented using equation 12 (Zhang, Yang and Başar, 2021):

$$\mathbb{E}\left[\sum_{t \geq 0} \gamma^t R(s_t, a_t, s_{t+1}) \,\middle|\, a_t \sim \pi(\cdot \mid s_t), s_0 \right] \tag{12}$$

With respect to this, the action value function, which is the Q-function, and the state-value function known as V-function can be defined under policy π using equation 13 and equation 14 (Zhang, Yang and Başar, 2021):

$$Q_\pi(s, a) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t R(s_t, a_t, s_{t+1}) \,\middle|\, a_t \sim \pi(\cdot \mid s_t), a_0 = a, s_0 = s \right] \tag{13}$$

$$V_\pi(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t R(s_t, a_t, s_{t+1}) \,\middle|\, a_t \sim \pi(\cdot \mid s_t), s_0 = s \right] \tag{14}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, the optimal policy is denoted by π^* and the values corresponding to it are referred to as the optimal Q-function and optimal state-value function.

RL algorithms are broadly classified into value-based and policy-based methods. Q-learning is a very popular value-based algorithm where the agent learns the optimal state-action value function estimate by constantly evaluating the Q-values (Tianxu Li et al., 2022). Similarly, Proximal Policy Optimization (PPO) is a policy-based algorithm which uses a penalty scheme where the clip function is implemented to keep a check on the deviation of new policy from the old one.

Temporal Difference (TD) learning is a model-free method where the value of sub-problems can be estimated with bootstrapping even if the estimation might be non-optimal sometimes.

The sub-problems represented by states in MDP are decomposed by TD learning with bootstrapping. One-step bootstrapping, denoted as TD(0), of a given value function $V : \mathcal{S} \to \mathbb{R}$ is defined using equation 15 (Huang, 2020):

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_t + \gamma V(S_{t+1}) - V(S_t) \right], \tag{15}$$

where TD target: $R_t + \gamma V(S_{t+1})$ and TD error: $R_t + \gamma V(S_{t+1}) - V(S_t)$.

The state-action pair derived using Bellman equation for the Q-values is given by equation 16 (Huang, 2020),

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \,\middle|\, S_t = s, A_t = a \right]. \tag{16}$$

where R_{t+1} is the immediate reward and $v_\pi(S_{t+1})$ is the discounted value of the subsequent state which captures the expected future reward.

An improved policy can be achieved by choosing actions greedily (equation 17):

$$\pi'(s,a) = \arg\max_{a'} q^\pi(s,a') \tag{17}$$

where the improvements can be ensured by the relation shown in equation 18 (Huang, 2020):

$$q_{\pi'}(s,a) = \max_{a'} q_\pi(s,a') \geq q_\pi(s,a) \tag{18}$$

The exploration of actions is done using an ϵ-greedy policy regardless of their Q-values but with a small probability ϵ. The Q-value of ϵ-greedy policy is given by equation 19 (Huang, 2020):

$$q_\pi(s,\pi'(s)) = (1-\epsilon)\max_{a\in\mathcal{A}} q_\pi(s,a) + \frac{\epsilon}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} q_\pi(s,a) \tag{19}$$

2.5.1 Q-Learning

Q-learning is an off-policy temporal difference control algorithm which reuses the past experiences. The one-step Q-learning follows an update rule as shown in equation 20 (Huang, 2020):

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\big[R_t + \gamma\max_{A_{t+1}} Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)\big] \tag{20}$$

where A_t is sampled by ϵ-greedy with respect to Q. One-step Q-learning updates the Q-values using only the immediate reward from the current transition. This could be less accurate when using function approximation to estimate Q-values. A better alternative is multi-step Q-learning, which considers rewards over multiple steps to refine the Q-value estimates. Since function approximation often introduces errors due to generalization, multi-step Q-learning must account for potential mismatches in the subsequent rewards to ensure that the Q-value function remains aligned with the expected return under the target policy.

In Q-learning, function approximation is used to estimate Q-values with some parameter θ. Then the update rule in equation 20 is reformulated as an optimization problem, where the parameter θ_t is adjusted to minimize the loss function $\mathcal{L}$, as given in equation 21 (Huang, 2020):

$$\theta_t \leftarrow \arg\min_\theta \mathcal{L}(Q(S_t,A_t;\theta), R_t + \gamma Q(S_{t+1},A_{t+1};\theta)) \tag{21}$$

The loss function can be Mean Squared Error (MSE), Mean Absolute Error (MAE), Huber Loss, etc.

2.5.2 Deep Q-Networks

The integration of Q-learning along with Deep Neural Network (DNN) gave an introduction to Deep Q-Network (DQN) and solution to many complex problems, particularly in high-dimensional single and multi agent decision-making. The optimal Q-learning action-value function approximated using DNN, is given by equation 22 (Abdul Mueed Hafiz and Bhat, 2020):

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{s=0}^{\infty}\gamma^s r_{t+s}\,\middle|\, s_t = s, a_t = a, \pi\right] \tag{22}$$

where r_t gives the maximized reward sum. The output layer of DQN has a separate output for each of the actions taken by the agent. Each of these outputs represent the predicted Q-value for that specific action in the given state.

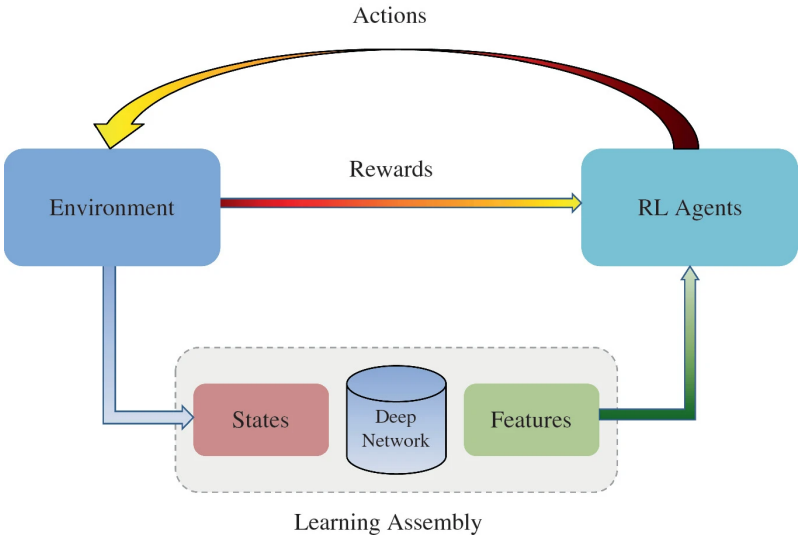


Figure 9: Structure of DQN (A. M. Hafiz, 2023)

The loss used in the learning update of DQN is defined as in equation 23 (Abdul Mueed Hafiz and Bhat, 2020):

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \tag{23}$$

with,

- γ is the entity discount.
- θ_i represents the DQN parameters for the i^{th} iteration.
- θ_i^- represents the DQN parameters for target computation for the i^{th} iteration. θ_i^- is updated with θ_i and is fixed between updates.

To perform *experience replay*, the agent stores its experiences (equation 24) and these are collected over time to form a dataset (equation 25) which represents the agent's past interactions.

$$e_t = (s_t, a_t, r_t, s_{t+1}) \tag{24}$$

$$D_t = [e_1, e_2, \ldots, e_t] \tag{25}$$

During learning, Q-value updates are computed using minibatches of experiences (s, a, r, s'), which are randomly and uniformly sampled from the stored experience collection D_t.

2.5.3 Double Deep Q-Networks

DQN max operator uses the same Q-values for action selection and evaluation, which can lead to overestimated values resulting in overly optimistic predictions (Abdul Mueed Hafiz and Bhat, 2020). To overcome this issue this problem, Double Q-learning was introduced. Here, the selection and evaluation processes are separated. Two Q-functions are learned with experiences randomly assigned to update either function. This creates two sets of weights, θ and θ'. For every update, one set is used for action selection (greedy policy determination), and the other for evaluating the selected action (value determination). This separation helps reduce overestimation bias in the value function with the use of Double Deep Q-Networks (DDQN).

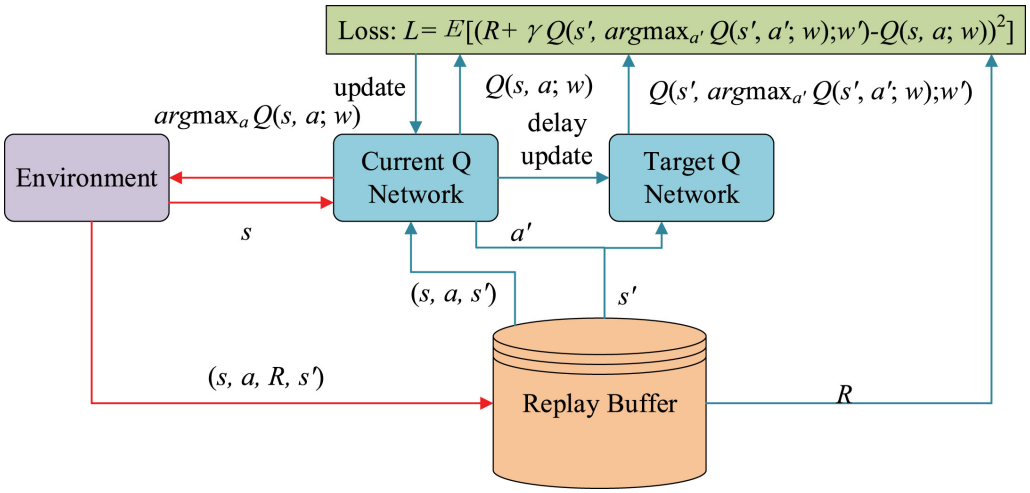


Figure 10: Framework of DDQN (Zheng et al., 2023)

The target for the Q-value update, with separated selection and evaluation components, can be mathematically expressed as shown in equation 26 (Abdul Mueed Hafiz and Bhat, 2020):

$$Y_t^Q \equiv R_{t+1} + \gamma Q(S_{t+1}, \text{argmax}_a Q(S_{t+1}, a; \theta_t); \theta_t) \tag{26}$$

Now, the Double Q-learning algorithm for the network can be written using equation 27 (Abdul Mueed Hafiz and Bhat, 2020):

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \text{argmax}_a Q(S_{t+1}, a; \theta_t); \theta'_t) \tag{27}$$

The value of the policy is evaluated using the second set of weights θ'_t, which can also be updated symmetrically by swapping the roles of θ and θ'.

2.5.4 Loss Function

The loss function in DDQN is normally calculated using MSE. It has certain limitations, for example when the error exceeds a value 1, it disproportionately emphasizes outliers by assigning them higher weights. This can harm the overall prediction accuracy of the model by reducing its performance on normal data points.

Mean Absolute Error (MAE) avoids overemphasizing large errors since it applies the same penalty regardless of error size. But MAE is not differentiable at zero and maintains a constant gradient even for small errors. This constant gradient makes it harder for the model to converge effectively and slows down learning.

To overcome these issues and improve the model, Huber loss function is used in this project work. The Huber loss blends the strengths of both MSE and MAE by applying MSE to small errors which ensures smooth convergence, and MAE to larger errors thus limiting the impact of outliers. The balance between the emphasis on MSE and MAE in the Huber loss function is determined by a threshold parameter. This balanced approach improves overall performance and learning stability. It is expressed as equation 28 (Zheng et al., 2023):

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{if } |y - f(x)| \le \delta, \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases} \tag{28}$$

where δ is the Huber loss parameter.

2.6 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) got introduced from the research conducted over the Multi-Agent Learning (MAL) approaches. The MAL research history can be divided into

two periods known as the startup and the consolidation period (Tao Li, Zhao and Zhu, 2022). Startup period research focused on how to apply Machine Learning (ML) techniques to overcome challenges in multi-agent systems. Distributed sensing, herd behavior, social learning, evolutionary computation, and artificial neural networks were explored in the study. This was not only a topic of interest for ML experts but also for researchers in field of game theory, economics, and biology. Their contributions helped to reshape concepts of equilibrium in games and provided new insights into evolutionary stable strategies. This period set the foundation for the broadening scope of MAL research.

During consolidation period, coinciding with the growing RL approach for single-agent scenarios, key studies sparked a wave of research focusing on extending RL techniques to multi-agent environments. Single-agent RL involves an agent learning to maximize its long-term rewards, which later was adapted to consider the interactions among multiple agents. To formalize these interactions game-theoretic models are essential, since they provided a quantitative way to understand how agents with different objectives and information coordinate their actions.

With no time, it became evident that challenges in multi-agent systems extended beyond simply applying single-agent learning algorithms. Strategic interactions between agents introduced new complexities, as agents not only rely on their own strategies and actions but also on the strategies of others. Since each agent lacks full knowledge of the decision-making processes of the others, they must form subjective models based on their observations. Each agent's learning process is influenced by these models, which in turn affects the overall learning outcomes. An agents' perceptions of their environment and the actions of others directly impact their decision-making. This highlights the importance of understanding the learning structure within MAL, which is critical for advancing the field further.

In competitive games, Multi-Agent Reinforcement Learning (MARL) involves multiple agents acting in the same environment to achieve individual goals (Zhang, Yang and Başar, 2021). This is also known as Adversarial Multi-Agent Reinforcement Learning (Adv-MARL). In these scenarios, the system's state evolution and the rewards each agent receives depend on the combined actions of all agents. Each agent aims to maximize its own long-term reward, which is influenced by the strategies of the other agents.

The tuple $(\mathcal{N}, \mathcal{S}, \{\mathcal{A}^i\}_{i\in\mathcal{N}}, \mathcal{P}, \{R^i\}_{i\in\mathcal{N}}, \gamma)$ represents a Markov Game which is a multi-agent extension of an MDP where,

- $\mathcal{N} = \{1, \ldots, N\}$ is the set of agents with $N > 1$.
- θ_i is the state space.
- $\mathcal{A}^i$ is the action space of agent i.
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition probability from $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ for any joint action $a \in \mathcal{A}$.
- $R^i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the immediate reward that agent i gets for making the transition from (s, a) to s'.
- $\gamma \in [0, 1)$ is the discount factor.

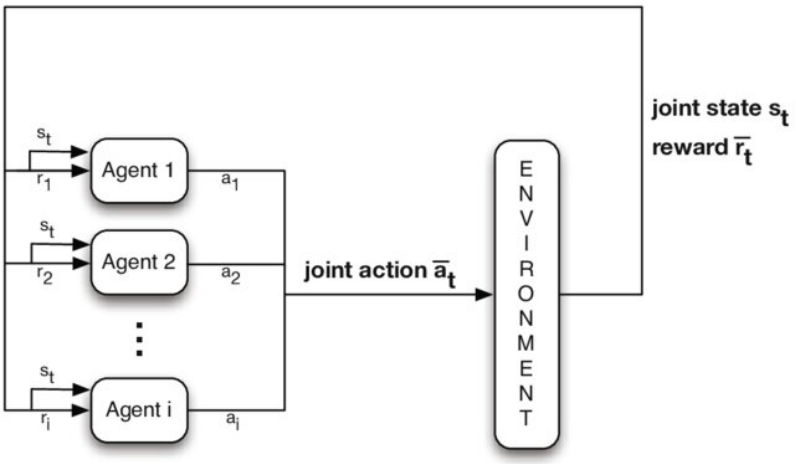


Figure 11: Multiple agents acting in the same environment (Nowe, Vrancx and De Hauwere, 2012)

3 Methodology

Safety is of prime importance in systems engineering. Reliability engineering makes it easier to understand, implement and assure safety in any system. Along with this, the maintenance of a system plays a crucial role in the long run of any industry, particularly in the automation systems. In that case, how efficient and thoughtful would it be if the risks and potential failure modes in a system could be identified at an earlier stage of development. If the potential failures in a system could be identified long before the failure itself, it can prevent a lot of failure cost for the industry along with potential risks to life and related systems. Failure of a component or set of components in a system is like a forest fire. It might start with a spark but can wipe out the whole vegetation in the area. Similarly, the faults in a system might start with basic event failure, which are the components present in the lower divisions of a system, that can result in the failure of connected components leading to the fault propagation through the system and finally causing system failure.

3.1 Experimental Description

The components in a system can be represented using a DFT which can further be utilized for risk analysis of safety critical systems. The aim of this project work is to use Machine Learning techniques to evaluate the potential risks in a system. The system is defined by a Dynamic Fault Tree prototype model. The basic events are the base level components of a system where the fault actually gets initialized. This fault then propagates to the next level of events/components which are joined through gates with or without condition(s). When certain conditions are met, the corresponding intermediate event fail. For example, when all child events of an intermediate event connected to child events via AND gate fails, then the intermediate event also fails. When such permutations and combinations align to the conditions of failure of intermediate events, multiple intermediate events can fail. These combinational intermediate event failures can lead to the top event failure condition resulting in system failure.

In this experimental setup, the DFT is defined in such a way that when the BE failure is triggered, the fault propagates through the system and causes a system failure. Similarly, when the failed BE is fixed or repaired, then the states of all the connected intermediate events which were in the failed state also get re-evaluated. Fault injection into the DFT is a critical step and doing it manually in a very complex system is not feasible. Thus, the incorporation of Adv-MARL approach into the system. Two adversarial agents, the red agent and the blue agent, will be initialized and trained over the DFT using DDQN algorithm. Here, the red agent will inject fault into the system, while blue agent will repair the events to fix the system. To facilitate the implementation, the experiments were conducted using an environment modeled after PettingZoo's multi-agent framework. PettingZoo provides a robust and flexible platform for multi-agent reinforcement learning, making it ideal for this study. The analysis of outcomes of the Adv-MARL games over the DFT would be the main focus of this project. And, how these outcomes can be used to re-model the DFT in such a way that the overall reliability of the system improves, resulting in a better fault tolerant system.

3.2 Project Architecture

An overview of the project implementation is shown in figure figure 12.

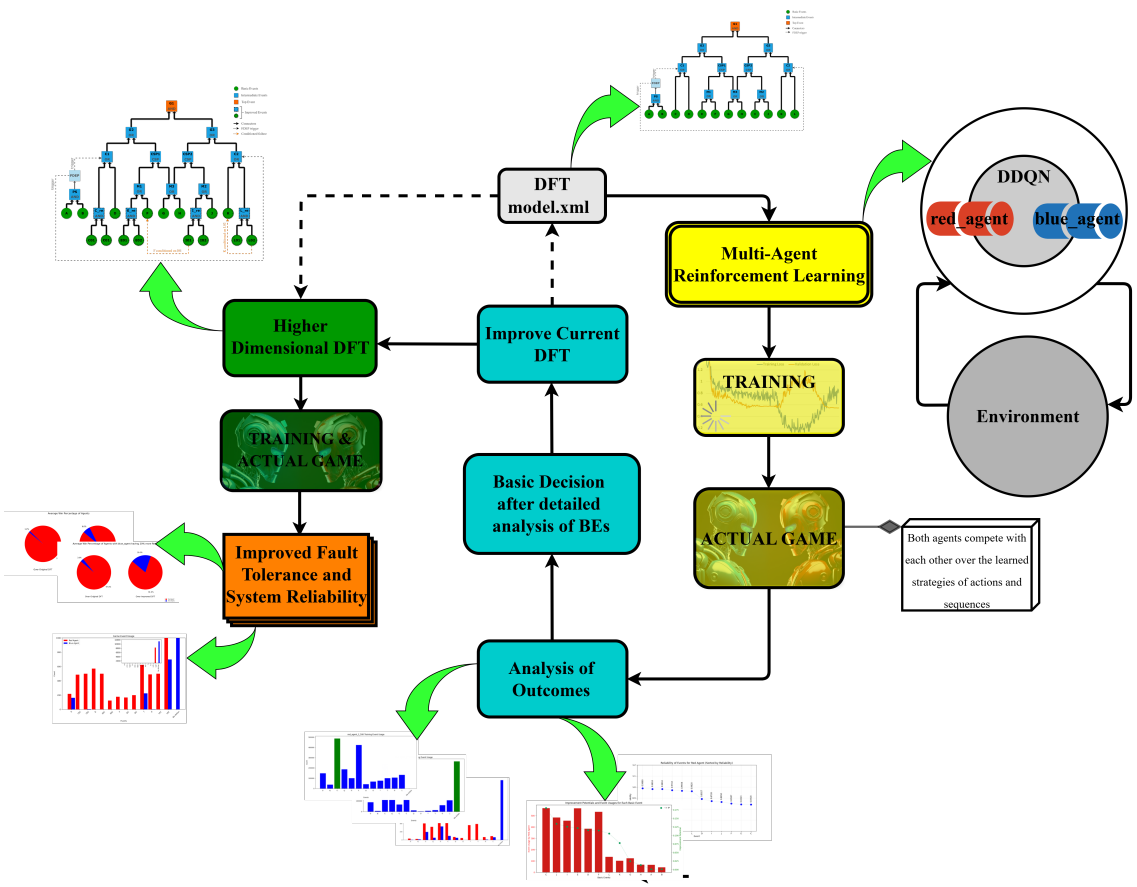


Figure 12: Overview of the Project

In simple terms, the analysis conducted in this project is based on the following summarized points:

- A test case DFT is created which represents a system.
- From the RL algorithms, DDQN is used to define the agent's algorithm. The agents are trained to compete with each other to inject fault into the system (red agent) and to repair the system (blue agent).
- After the training, the actual game is played between the red and blue agent, and the respective results are analyzed to form a basic decision.
- While implementing the basic decision, the BEs are ordered according to their improvement potential and reliability metrics. Also, how these events can be modeled to improve system reliability is another question.
- Once all these are finalized, the initial DFT is modified accordingly and the RL model is re-trained over the new data to see the results.
- Actual game is played again over the re-trained model and the outcomes are analyzed to verify if the expected improvements are met or not.

This is an overview of the implementation of this project. A detailed study and discussion will be done in the upcoming sections.

3.3 Implementation of Adversarial Multi-Agent RL games in the DFT Environment

In the modeled DFT, the implementation of agent interaction is through a zero-sum game approach to influence the system's functionality. The design ensures that only BEs can be directly accessed by the agents to cause failure and correction, with all other event states being derived from these actions. The system also includes methods for managing the state of events, tracking real-time changes and allowing the game to reset when certain conditions are triggered. Additionally, the game allows the agents to adjust their strategies based on their resources and the evolving state of the system. One such strategy is the "No Action" step, where the agent can skip an action if it learns that it is the best thing to do at that moment.

The game is built using PettingZoo library to create a turn-based environment where agents make decisions based on the state of the system. The states of the events are initialized as 1, indicating that the system is fully operational at the beginning. By taking some actions the agents interact with the environment that change the states of the BEs, either by injecting faults (by red agent) or repairing them (by blue agent) when it is already in a failed state. The red agent takes the first action in a game sequence since the events are fully functional at the beginning, followed by an action by the blue agent. The environment provides observations in the form of binary vectors representing the states of each event. And, the agents are rewarded based on the changes they cause. The system tracks the rewards and penalties associated with the top event state, with red agent receiving a reward when the system fails and the blue agent getting rewards for keeping the system operational throughout a game.

The reward is structured as:

- Red Agent: Executes strategic fault injection to maximize own reward function. Red agent receives a reward value of +10 for causing a system failure and a penalty of -1 for not able to fail the system in the given time steps.
- Blue Agent: Learns strategies to effectively fix the faults injected by red agent. blue agent is highly rewarded, a value of +100, for keeping the system functioning until truncation and also gets punished by a value of -10 for losing the game.

To support the game-play, the system uses several key functions, including action masking, which ensures that agents can only perform valid actions based on the current state of the system. If an event is already in a failed state, then that particular event cannot be accessed by the red agent until it gets repaired and becomes completely functional again. The environment's step function processes each action of the agents. Then, updates the system state and adjusts the rewards accordingly. This setup enables the agents to optimize their strategies over time, helping them to learn how to act effectively within the constraints of the game environment. Ultimate goal of each agent is to maximize their rewards as discussed in the literature section 2.5.2. This is strategically achieved either by failing the top event or keeping it operational by red agent and blue agent respectively, until the game terminates once the top event fails or the maximum number of steps has reached.

The Q-learning algorithm Double-DQN used for policy optimization incorporates a ReplayBuffer to store and sample training data. It employs a DoubleDeepQNetwork class with separate actor and critic networks. The architecture includes two hidden layers with 512 neurons each with Leaky Rectified Linear Unit (Leaky ReLU) activation function. The loss is calculated based on SmoothL1 loss, also known as Huber Loss. Each agent has two networks, current and target policies, and selects random actions based on a decreasing exploration rate. Once the exploration is completed or have reached the minimum value, the agents exploit the actions based on the learned policies. The learning process involves updating policies with sampled data and target network parameters are periodically refreshed. The game is played over multiple episodes while training and in this implementation the training was done over 100,000 training epochs. The model parameters are saved once the training is completed and are used for testing with the learned strategies in the actual game scenario of which the outcomes are analyzed and certain decisions are made in order to improve the system reliability.

4 Evaluation Concept

4.1 Test case DFT

The DFT in this project is provided as an XML file which contains the basic, intermediate and top events along with their parameters and the structure of the represented system. Each basic event is described by parameters such as Mean-Time-To-Repair (MTTR or mttr), repair cost and failure cost. The structure of the DFT is defined by Precedence class using source, target and competitor information stated in the XML file. The precedence helps to capture temporal dependencies, sequence and/or failure order accurately, making the model a reflection of real-world system. The test case DFT is as shown in figure 13, with name of the event and type of gate.

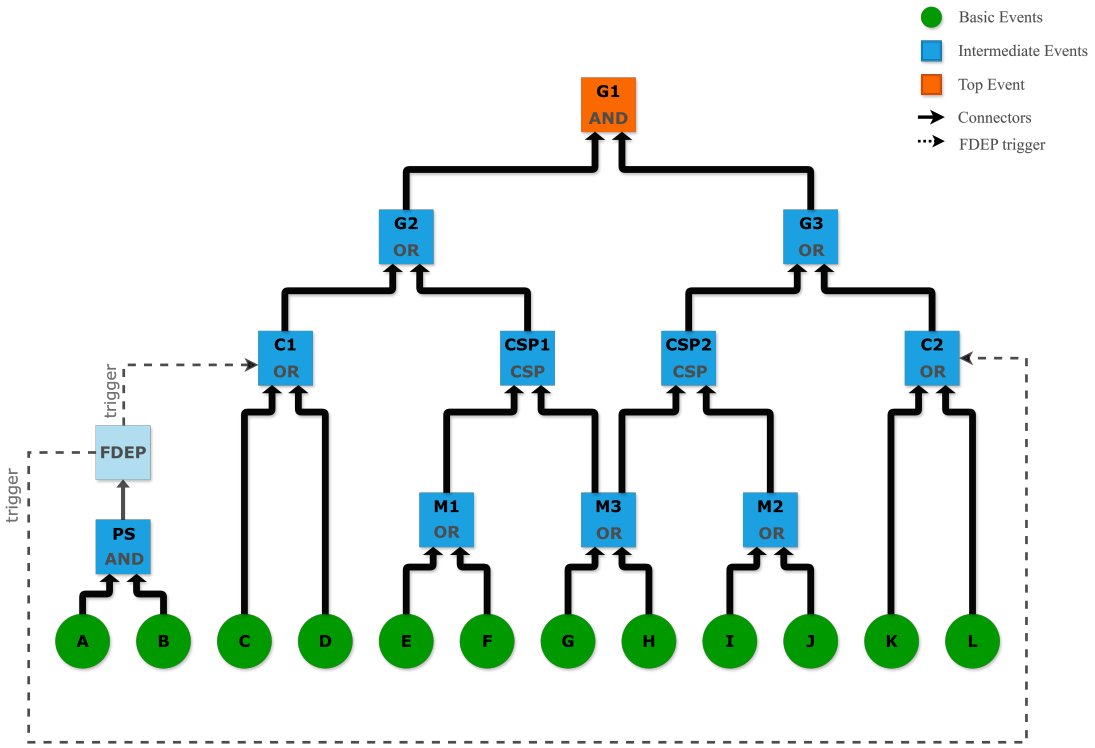


Figure 13: Lower Dimensional DFT

DFT shown in the figure is the initial DFT model or the Lower Dimensional DFT (LD_DFT). The failure or working state of events are represented using 1 or 0. A “state = 1” depicts the component is functional while “state = 0” stands for faulty component or system. Basic events are represented using the green circles named from A to L, each having specific values of mttr, repair cost and failure cost.

- Mean-time-to-repair: mttr is the representation of average time it takes to repair a basic event when it is in a failed state. For example, an mttr value equal to 3 indicates that it will take 3 time-steps for the event to get repaired when the blue agent chooses to repair it in a game.
- Repair cost (”repair_cost”): The cost incurred to repair a BE when it has failed. This cost is reduced from the resource allocated to the blue agent each time it attempts to repair a failed event.
- Failure cost (”failure_cost”): The cost incurred to cause a failure to a BE. Here, the cost is subtracted from the resource allocated to the red agent each time it attempts to fail an event.

The parameters of the BEs are as given in table 2:

Basic Events	MTTR	Repair Cost	Failure Cost
A	3	1	3
B	7	3	8
C	2	1	2
D	4	4	5
E	3	7	6
F	2	3	3
G	3	7	6
H	2	3	3
I	3	7	6
J	2	3	3
K	2	1	2
L	4	4	5

Table 2: Basic Event Parameters of LD_DFT

Intermediate events are depicted using blue squares with name on top and the type of gate below. As known from the basics, for the AND gates to fail both the input must be in a failed state. For OR gates, failure of any of the child events results in the failure of OR gate. Now, lets see how failure of intermediate event can cause system failure.

- If both A and B fails, PS fails which triggers the FDEP gate. This results in the failure of both C1 and C2. The fault further propagates to G2 and G3, causing G1 failure ($system.state == 0$).
- Failure of C1 and C2 through different combinations can create a system failure.
- M1, M2 and M3 are connected to the CSP1 and CSP2 events via a 2-out-of-3 setup. If minimum two events out of the three, M1, M2 and M3 fails, then both CSP1 and CSP2 fails. Fault further moves to the parent events G2 and G3 causing system failure.

These are the combinations through which red agent can cause a system failure in the given DFT.

4.2 Evaluation Metrics

As mentioned in 3.2, once the training is completed over the lower dimensional DFT shown in figure 13, the actual game is played using the trained weights. The outcomes of these games need to be extracted and analyzed to develop certain decisions with a focus on improving the reliability of the system. The developed concept needs to be implemented and tested to verify the success of the implementation. Certain assumptions declared before the actual implementation of the concept:

Assumption 1: The DDQN model implemented in this project is assumed to give the best performance in the current environment. There are certain limitations concerning the performance of the agents, improving which is out-of-scope of this project. Since, outcomes of the improved RL model estimations are satisfactory and serves the purpose of this project work, the focus remains on using this model rather than enhancing its performance. Therefore, the current implementation of the DDQN model will be used without further modification for the analysis of outcomes and decision-making.

Assumption 2: The reliability and IP calculations conducted in this project are based on the *red_agent's* policy. Event usage by the *red_agent* gives an estimate of failure rates of those events. It refers to instances where the *red_agent* selects an event, which signifies an attempt to inject fault into that event. Consequently, the term "event usage by red agent" reflects the number of times an event has experienced a failure due to the *red_agent's* actions.

The evaluation metrics are as follows:

1. **Number of games won:**
   - An overall idea about how good the strategies and learning of the *red_agent* and *blue_agent* can be measured using the number of games each agents wins in a set of games played.
   - This also partly indicates the how reliable the given system is.
   - The effectiveness of the changes made to DFT for the evaluation of reliability can be indicated using the number of games the agents win and loose.
2. **Event Usage by *red_agent*:**
   - The number of times an event is used by *red_agent* indicates the frequency of failure of that event in a game or across multiple games.
   - This data is used to estimate the most frequently targeted BEs, which are identified as critical events because they contribute the most to system failures.
   - Comparison of event usage between low dimensional DFT and high dimensional DFT provides an insight into how much the reliability of events have improved under the *red_agent's* policy.
3. **Critical Event Analysis:**
   - Analysis of critical event(s) to understand which BEs are the most frequently attempted ones by the agents during the game.
   - Finding the most frequent event helps in improving fault tolerance, either by eliminating the event or by making it more resilient to failures.
   - The latter can be achieved by improvising the model using two more basic events connected to the critical event through an AND gate. This change in the model requires the *red_agent* to fail both new Basic Events to make the critical event failure, thereby increasing the difficulty of causing a system failure.
4. **Action Sequence Analysis:**
   - The sequence of actions taken by each agent is analyzed to determine the patterns or sequential strategies that are implemented by the agents, which helps them gain the maximum rewards and also win the game.
   - This could be a very useful method to understand the depth of an agent's decision-making which results in an improvised model to improve the fault tolerance.
5. **Improvement Potential (IP) Analysis:**
   - IP indicates the importance of the events with respect to the system's reliability.
   - Improving an event with higher values of IP can impact the reliability of the system, potentially leading to improved performance or unintended vulnerabilities, depending on how these changes interact with the overall system dynamics.
   - The events with the least value of the IP has less impact on the overall system reliability.
6. **Reliability Analysis:**
   - Indicates how reliable an event is. The higher the value of reliability of an event proves that the event is difficult to fail or fails less when compared to other events with lower values of reliability.
   - The improvement potential and reliability are inversely proportional. The event with higher IP has lower reliability.

4.3 Improvement Potential Calculations

The choice of IP as an importance measure for the analysis was concluded compared to the other available importance measures, since it explicitly considered the actual reliability or failure rate of the components, which feels more realistic and practical. Other measures, like Birnbaum's measure, relies on ideal scenarios of perfect functioning and complete failure of components. Whereas, IP takes the current state of a component, which makes it more applicable for prioritizing improvements and conducting maintenance strategies in real-world scenarios.

In this project work, the formulas discussed in 2.3.4 were slightly modified for the calculations, to adjust to the available parameters and metrics. As mentioned in the assumptions, the calculations are based on the red agent's policy. Thus, whenever event usage is mentioned in the formulas, it indicated the fault injection by red agent over that event.

Step 1: Calculate System Failure Rate

$$\text{System Failure Rate} = \frac{\text{Number of Wins by Red Agent}}{\text{Total Number of Games Played}} \tag{29}$$

Step 2: Calculate Total Event Usage

$$\text{Total Event Usage} = \sum(\text{Event Counts Triggered by the Red Agent}) \tag{30}$$

Step 3: Calculate BE Failure Rate

$$\text{Failure Rate Event}_i = \left(\frac{\text{Usage of Event}_i}{\text{Total Event Usage}}\right) \times \text{System Failure Rate} \tag{31}$$

Step 4: Calculate Intermediate and Top Event Failure Rate

Based on the basic event failure rates, propagate the failure rates through the intermediate events, considering the gate type, to calculate the top event (G1) failure rate. A detailed calculation is given in the appendix.

Step 5: Calculate Improvement Potential

The calculated G1 failure rate is stored and is called the *Original Top Event Failure Rate*. Then, for each BE, its failure rate is set to zero, and the failure rates of all dependent intermediate and top events are recalculated. This is known as the *Recalculated Top Event Failure Rate*.

Now let's calculate the improvement potential (IP) for each BE:

$$\text{IP}_i = \text{Original Top Event Failure Rate} - \text{Recalculated Top Event Failure Rate}_i \tag{32}$$

Step 4 is repeated to calculate IP for each of the basic events.

Step 6: Calculate Reliability

$$\text{Reliability of Event}_i = 1 - \left(\frac{\text{Usage of Event}_i}{\text{Total Event Usage}}\right) \times \text{System Failure Rate} \tag{33}$$

$$\text{System Reliability}, R(t) = e^{-\lambda t} \tag{34}$$

where t is the time and it is assumed to be 1.

5 Analysis

The implementation of this project follows the OOPS paradigm and is carried out using python programming language, PyTorch libraries and PettingZoo AEC environment. When the training is completed, models are saved and is used for the actual game. The analysis is done based on the outcomes of actual game.

To ensure reproducibility of results, a fixed seed is set across all random number generators used in the implementation. This includes Python's built-in random module, NumPy, and PyTorch. The seed ensures that every run of the training and testing processes produces consistent outcomes, to facilitate reliable comparison between experiments. For GPU computations, PyTorch's deterministic settings are enabled to control the behavior of operations dependent on parallel processing. This guarantees that the randomness involved in model training, such as weight initialization and action selection, does not introduce variability into the results.

5.1 Event Usage Analysis

The training process extends over 100,000 epochs. Initially, the agents explore the environment by taking random actions. This exploration is guided by a controlled exploration rate, which gradually decreases to a minimum value as training progresses. Over time, the agents learn to adopt the optimal policy that maximizes rewards and improves their chances of winning the game, based on insights gained during exploration. Eventually, they exploit these learned actions and sequences to achieve maximum rewards. Game terminates when red agent wins the game or when truncation condition occurs, that is, the maximum step has reached.

Training Event Usage Plot:

The win percentage of red and blue agents during training was 94.41% and 5.59%, respectively.

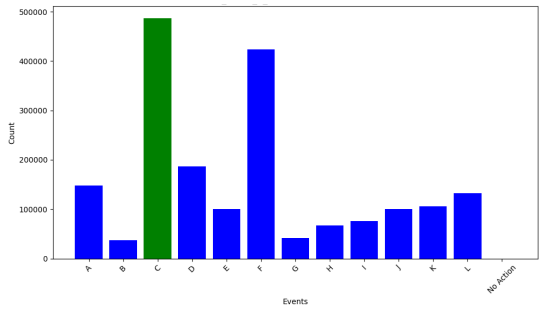


Figure 14: Red Agent Event Usage

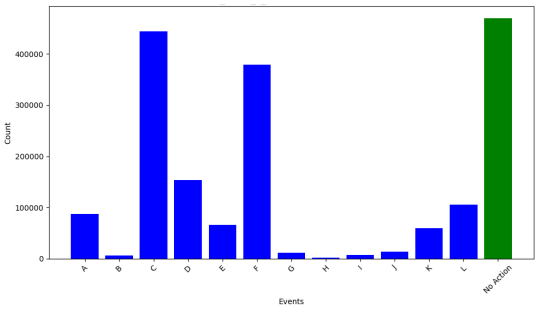


Figure 15: Blue Agent Event Usage

After training, based on the learned strategies the actual game is played, where the game ends when the maximum step limit is reached. The game is initialized with the step limit and and the agents should make their strategic moves within this limit to win the game.

Actual Game Event Usage Plot:

For the analysis, the model was evaluated by playing games consisting of 500, 5,000 and 25,000 rounds. The results across these sets of games were consistent. The pattern and proportion of the event usage remained the same, only the absolute numbers changed depending on the number of games played in each set. For the purposes of reliability and IP analysis a game of 500 rounds was used. In the actual testing phase, each game concludes when the maximum number of steps is reached. At this point, the system's state is recorded as the final state for that game. The plots for 500 and 5000 rounds of games are as shown in figure 16 and 17 respectively.

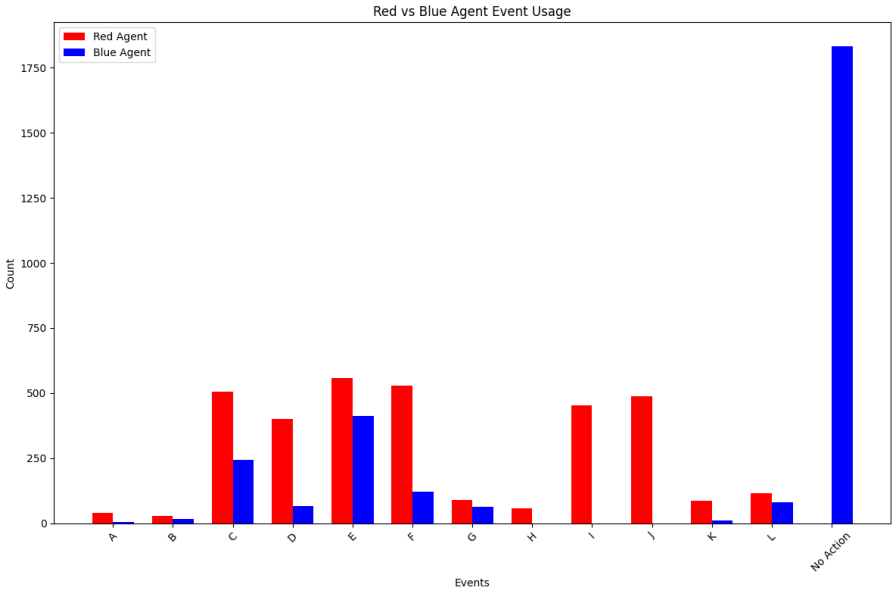


Figure 16: Event usage over 500 games

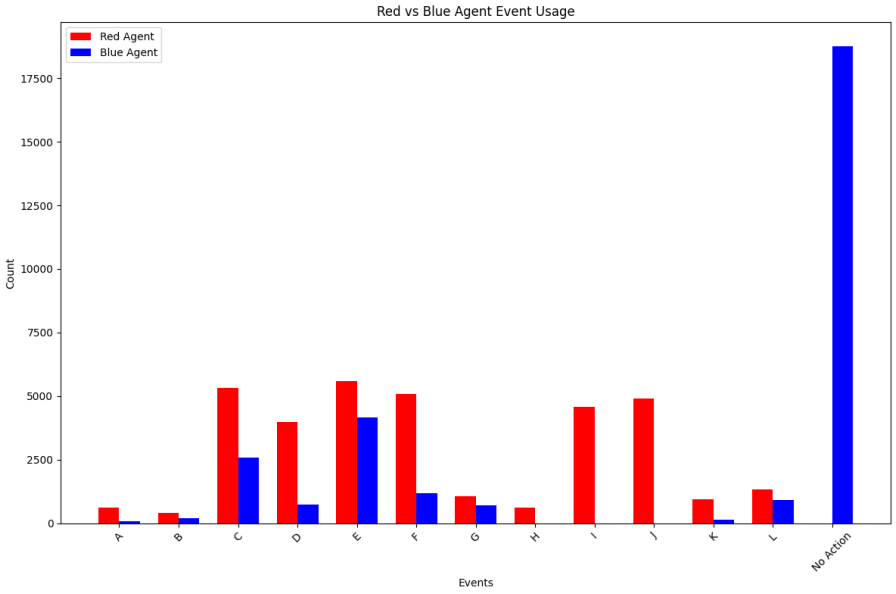


Figure 17: Event usage over 5000 games

Inferences from the game event usage outcomes:

- From the actual game plots it is very clear that the red agent frequently uses the events C, D, E, F, I, and J.
- The combination of A and B also triggers system failure through the FDEP gate condition as discussed in 4.1. The failure cost of event B is high and failing both A and B might become expensive for the red agent. Also, the significance of these BEs with respect to the number of times the fault is injected through them is fairly low. Thus, A and B are not considered as critical events.
- The combination of (E or F) with (I or J) fails the 2-out-of-3 intermediate events where the fault propagates through CSP1 and CSP2 to G2 and G3, causing system failure : G1 system

state equal to 0. Also, (G or H) combinations along with (E or F) or (I or J) also causes system failure, but their contribution is very compared to the other combination. Thus, are not considered as critical.

- Combination of (C or D) along with (K or L) triggers fault from both sides of the DFT resulting in system failure.
- Comparing with the training event usage plot for the red agent and blue agent, the events I and J are failed good number of times by red agent but are not fixed that many times by the blue agent, rather the blue agent takes "No Action" more often. This behavior allows the red agent to exploit a strategic advantage during the actual game. By repeatedly targeting events I and J, the red agent creates failures on one side of the DFT, knowing that these events are unlikely to get fixed by the blue agent.

An average winning percentage for the red and blue agents are estimated to be 99% and 1%, respectively. Win percentage for both agents for different game sets are given in figure 18:

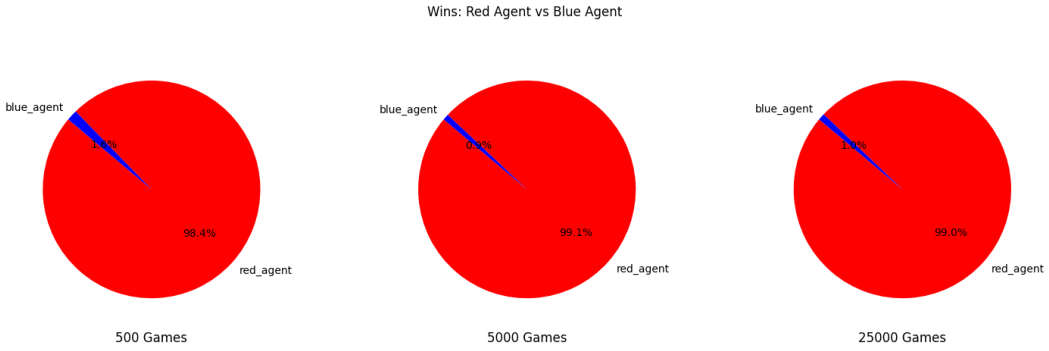


Figure 18: Game Win Percentage

Event Sequences: The most used event sequences are given in the table 3.

Most frequently occurring sequences	**('F', 'D', 'C', 'E', 'J', 'I')**	**('F', 'D', 'E', 'J', 'I')**
Reward red_agent gets by taking this sequence of actions	24	23
Game status	won	won
System state during the actions	(1, 1, 1, 1, 0, 0)	(1, 1, 1, 0, 0)
Occurrence of the sequence in 500 games	114	134
Occurrence of the sequence in 5000 games	1227	1356

Table 3: Critical BE sequence - Status

Note: Blue agent does not take relevant actions and mostly chooses "No Action". It chooses an action to repair the events when the system failure occurs, until then it chooses to take "No Action". Or, sometimes it taken an action to repair when an events fails but after that it waits until the system failure occurs. Meanwhile, it takes only "No Action". This is a major drawback, but still the main objective of this project is to verify if the system reliability improves or if the blue agent wins more games by improving the critical basic events.

Conclusion from 5.1

Basic events C, D, E, F, I and J are the critical events base on the actual game (test) outcomes. These events can be modified to see if the system fault tolerance improves or not.

5.2 Improvement Potential Analysis

Analysis of outcomes of the test games have proved that the events like C, D, E, F, I and J are the critical events. Lets further analyze the events using the importance measure Improvement Potential.

Using the formulas stated in 4.3, the IP of the events were calculated with respect to red agent's policy. The calculation was based on 500 rounds of games. This is graphically represented in figure 19:

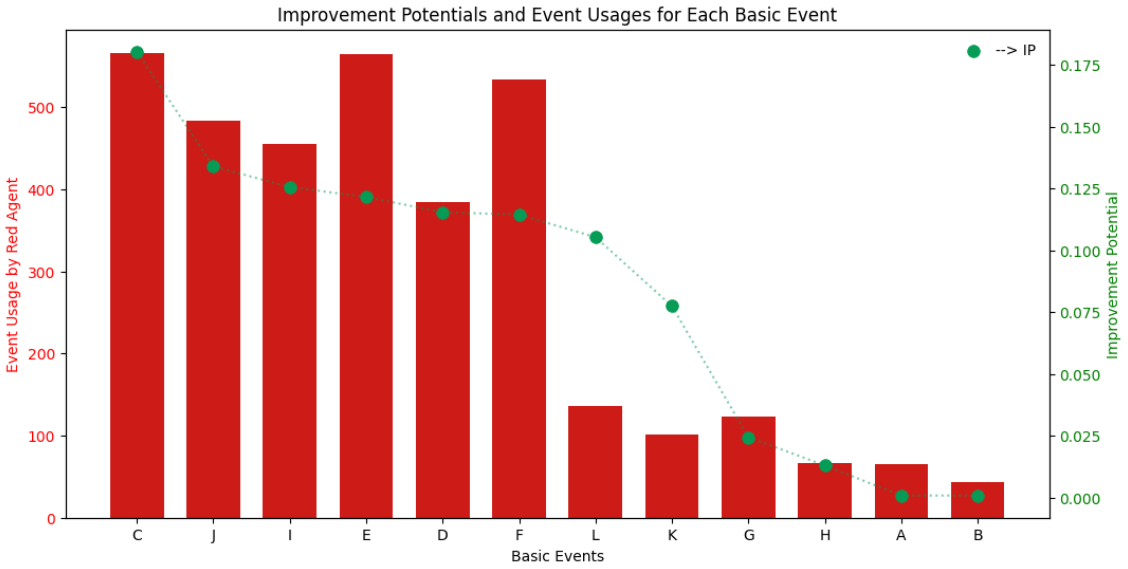


Figure 19: Improvement Potential of BEs based on Red Agent's Policy

Inferences from IP graph:

- As discussed in 5.1, events C, D, E, F, I and J are are once again proved to be critical from the IP analysis.
- From the IP value, C is the most critical BE, followed by J, I, E, etc.
- Event D has a slightly higher IP value than F even though F has failed more number of times than D. This difference arises due to the position of event D in the DFT. Failure of D alone can easily trigger a fault from the left side of the DFT causing G2 to fail, but the failure of F alone cannot cause a fault propagation till that level as D can. Only a failure of F in combination with (G or H) or (I or J) can cause a failure at that level or higher. Thus, while calculating IP, D has higher importance than F.
- K and L, which has been used less frequently by the red agent, still has significantly high IP values. This is also due to their position in the DFT, as failure of either of K and L can cause a fault propagation till G3 level.
- G and H is connected to the spare gate in the 2-out-of-3 gate, plus, has significantly low failure rates. Thus, they are not critical in this analysis procedure even though the failure rate of G is higher than K. The event usage of L and G is close enough, still the IP value is largely apart. This is the significance of position of events in a DFT, or to be clear, the importance of role of a component in a system.

- For the events at the similar position in the DFT, for example, C and D or K and L, the IP value is proportional to the failure rate of events.

Note: Event usage analysis relies on the red agent's strategies and event parameters. On the other hand, IP analysis relies on red agent's strategies and DFT structure. Red agent's strategies also depend on the DFT structure. The objective of the 3rd point in the above inference is to point out that IP analysis assigns slightly more significance to DFT structure along with all other aspects involved.

Reliability of Basic Events

Based on the event usage by the red agent, a rough calculation for the reliability of the BEs was done. The reliability chart for the BEs is as shown in figure 20:

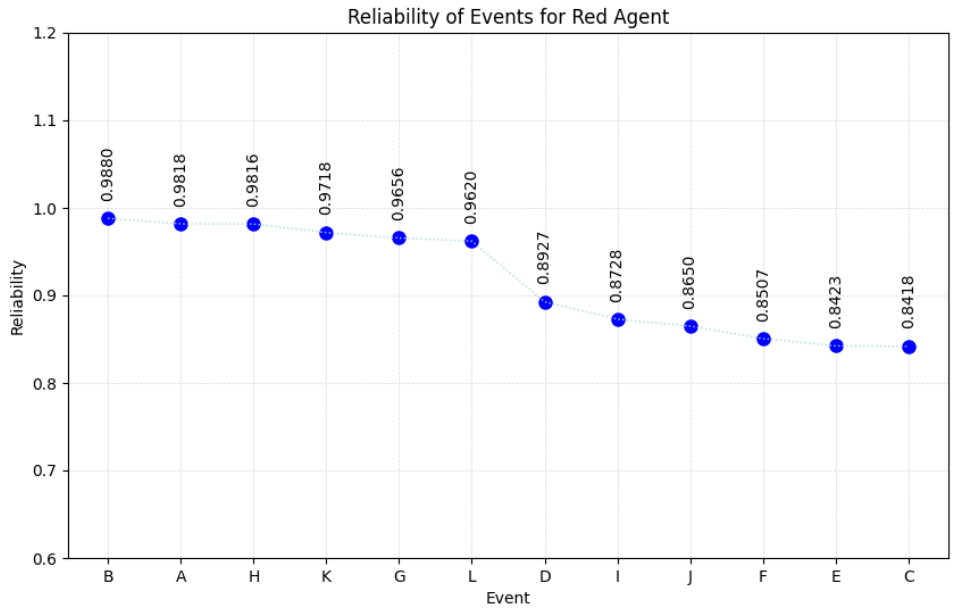


Figure 20: Reliability of BEs based on Red Agent's Policy

Inferences from reliability graph:

- The reliability of events are inversely proportional to the improvement potential.
- More reliable events or components have less potential for improvement and are therefore given less importance. Conversely, less reliable events have more contribution to the overall system reliability if modified properly or made redundant.

Note: How system reliability have changed/ improved has been plotted and discussed in the results in the upcoming section.

Conclusion from 5.2

Improving the DFT based on 5.1 would still leave the red agent to choose K and/or L to cause fault from right side of the DFT. Thus, along with critical events from event usage analysis, K and L have been concluded to be critical as well based on the IP analysis.

5.3 Basic Decision to modify Initial DFT

Changing the structure of the DFT especially by adding redundancy to improve reliability is an effective way to do the analysis of system reliability improvements. The following (Table 4) are the improvement strategies which will be implemented and tested for reliability analysis of the system and the events in the system.

Based on the analysis conducted in 5.1 and 5.2, the critical events are C, D, E, F, I, J, K and L.

Basic Events	Changes/ Additions	MTTR	Repair Cost	Failure Cost	Initial Value(s)
C	C01	2	1	2	C ⇒ 2, 1, 2 New values same as C
	C02	2	1	2	
D	-	MTTR reduced: 4 to 2	Repair cost reduced: 4 to 3	-	-
E	E01	2	6	5	E ⇒ 3, 7, 6
	E02	3	4	4	
F	Conditioned on I01	-	2	9	F ⇒ 2, 3, 3
I	I01	2	4	3	I ⇒ 3, 7, 6
	I02	2	3	3	
J	-	-	-	Failure cost increased: 3 to 8	-
L	L01	2	2	3	L ⇒ 4, 4, 5
	L02	2	3	3	
K	Conditioned on L02	-	-	-	-

Table 4: Modification to LD_DFT

Some key insights about the modifications:

- Changes are made only to the basic events. There are no changes to the intermediate events and their connections.
- Events C, E, I and L are made redundant by introducing two more new child BEs under each of them, connected via an AND gate. Therefore, C, E, I and L are the new parent events and is considered as intermediate events.
- The redundant events C, E, I and L are denoted as C_re, E_re, I_re and L_re.
- For events D, F and J, the parameter values have been modified to see how the agents will act to the changes. Events have been made costly to fail and easy to repair.
- K and F are made conditioned up on other events. F is conditioned on I01 such that if the red agent wants to inject a fault into F, it must ensure that I01 is already in a failed state. Otherwise, F cannot fail. Similarly, K is conditioned on L02.

6 Results and Further Discussions

The model training using new DFT, which is also known as Higher Dimensional DFT or HD_DFT, is conducted without any changes to the RL model used for LD_DFT. Changes are only made to the DFT itself to make it redundant and fault tolerant.

The HD_DFT is depicted in the figure 21.

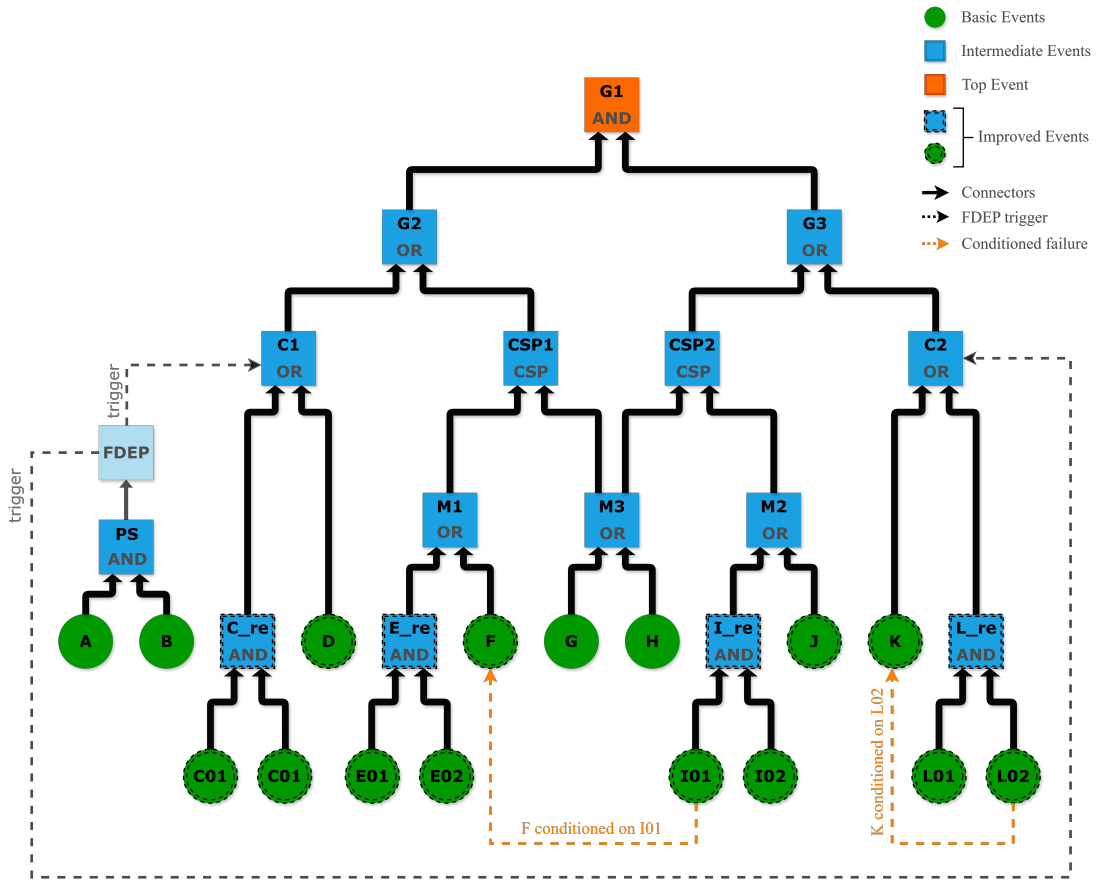


Figure 21: Higher Dimensional DFT

The parameters for the HD_DFT is shown in table 5:

Basic Events	MTTR	Repair Cost	Failure Cost
A	3	1	3
B	7	3	8
C01	2	1	2
C02	2	1	2
D	2	3	5
E01	2	6	5
E02	3	4	4
F	2	2	9
G	3	7	6
H	2	3	3
I01	2	4	3
I02	2	3	3
J	2	3	8
K	2	1	2
L01	3	2	3
L02	2	3	3

Table 5: Basic Event Parameters of HD_DFT

6.1 Event Usage

Training Event Usage Plot for HD_DFT: A training for 100,000 training epochs was conducted again and it was used for the actual game or testing. The win percentage of red and blue agents were 77.57% and 22.43% respectively.

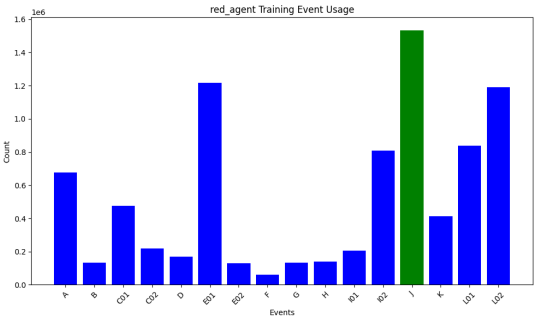


Figure 22: Red Agent Event Usage

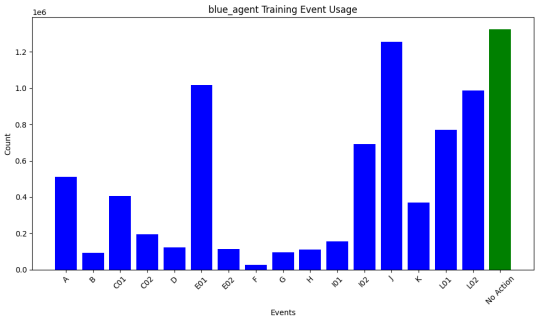


Figure 23: Blue Agent Event Usage

Actual Game Event Usage Plot for HD_DFT: The game was tested with equal resources for both agents. An actual game for 500 and 5000 rounds were played. The pattern still remains the same and the blue agent strategies are week.

The results are as follows:

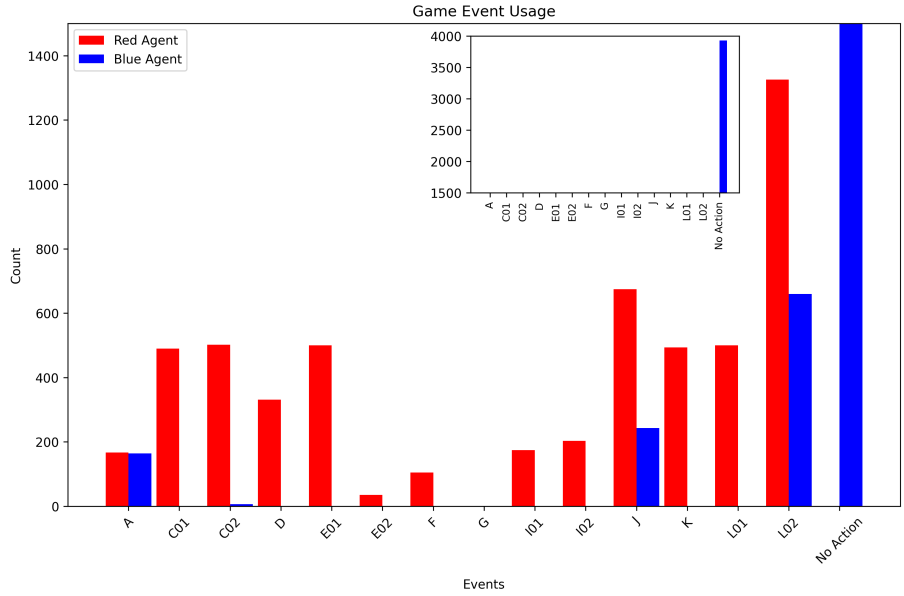


Figure 24: Event usage over 500 games

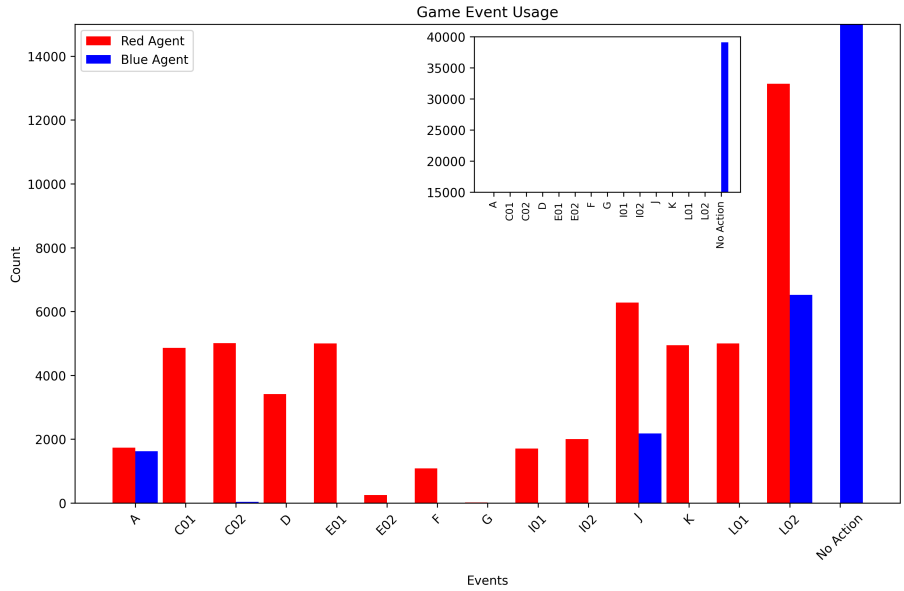


Figure 25: Event usage over 5000 games

Inferences from the game event usage outcomes of HD_DFT**:**

- The number of times event C fails in LD_DFT is approximately equal to the number of times event C_re fails in HD_DFT. While the failure count is similar, the cost of failure of C_re is double in HD_DFT. These redundancies make it more challenging for the red agent to cause system failure due to limited resource availability.
- The red agent's selection of event D has decreased in the new DFT compared to the old DFT. Minor improvements made to event D in LD_DFT have resulted in a slight reduction in its failure numbers in the new DFT, highlighting the proportional relationship between modifications and failure reduction.
- Fault injection via E_re has become difficult due to the failure cost of E01 and E02. Consequently, the red agent tends to choose events with low MTTR and high repair costs between

E01 and E02, aiming to have them fixed quickly but at a high expense to the blue agent. This strategy allows the red agent to deplete the blue agent's resources and win games strategically. Even though this is a strategic action, it cannot initiate significant number of fault propagation in the DFT because of low failure rate of E02. Adding reliability to the system.

- The failure rate of F has significantly decreased because its failure cost was set very high, and it was conditioned on I01.
- The combined parameters of I01 and I02 were set to match those of I (5.3), although I has a slightly higher MTTR value. As a result, the total failures of I01 and I02 are slightly lower than the failures of I. Since, I01 and I02 are connected to I_re through an AND gate, the failures of I_re is slightly less than half of I, demonstrating the impact of redundancy in system components.
- K was only made conditioned on L02. If L02 is in a failed state, K is highly vulnerable to failure. This is reflected in the failure numbers of K as depicted in the actual game plots.
- While the failure costs of L01 and L02 are the same, L02 has a higher repair cost and a lower MTTR. This explains why the red agent chooses L02 more frequently than L01, as it helps exhaust the blue agent's resources. However, the failure of L_re depends on the less frequently failed event L01 due to their AND gate connection.
- When comparing the old and new DFTs, among all the modifications, K, L01 and L02 are relatively more vulnerable to failure. As a result, their event usage has increased significantly.
- Compared to other events B is very expensive for the red agent. Therefore it was not even used a single time in the actual game.

6.2 Win Percentage

Test 1: Initially both the agents were tested by allocating equal resources. The resulting win percentages of the red agent and blue agent is given in figure 26.

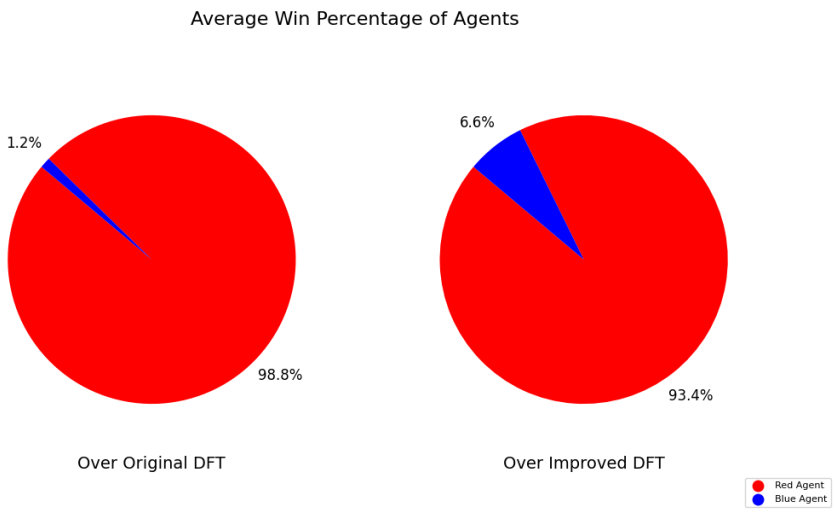


Figure 26: Game Win Percentage Comparison with Equal Resources

Test 2: The resources of the agents were slightly modified for a second test. The blue agent was given 20% more resources than the red agent. The results are as follows:

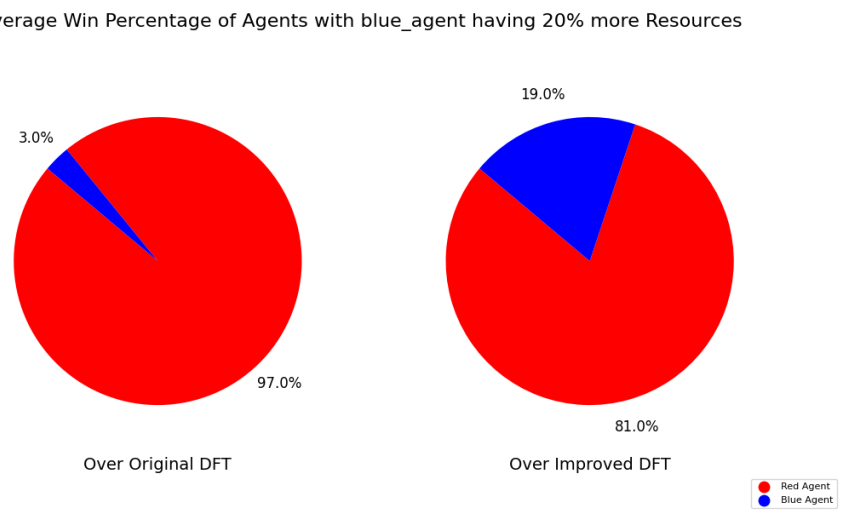


Figure 27: Game Win Percentage Comparison with Unequal Resources

Inferences from win percentage:

- The win percentage of blue agent has improved under the redundant DFT, indicating that system fault tolerance can be improved by enhancing the low level components in the system.
- The increase in game win percentage for the blue agent over the modified DFT demonstrates that the Adv-MARL framework can effectively identify critical components in a system, and improving these critical events enhances overall system reliability.
- Allocating additional resources to the blue agent in a meaningful way significantly increases its chances of winning the game.

6.3 System Reliability

System reliability was calculated for the initial and improved DFTs.

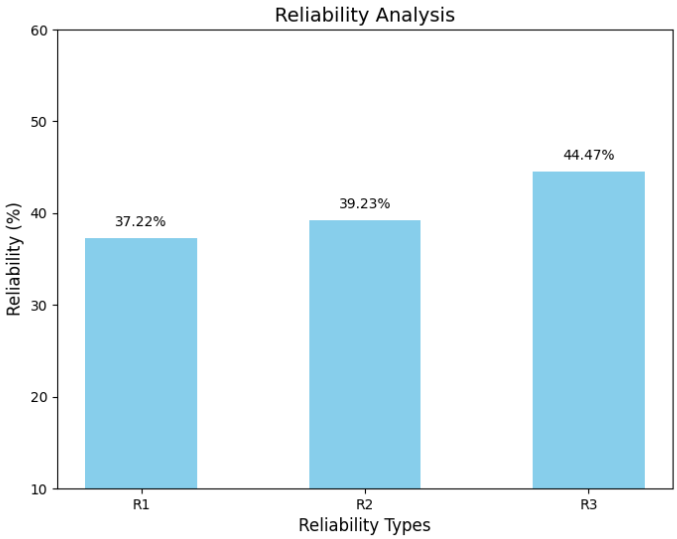


Figure 28: System Reliability Chart

where,

- R1: System Reliability modeled using LD_DFT.
- R2: System Reliability modeled using HD_DFT.
- R3: System Reliability modeled using HD_DFT with more resources for *blue_agent*.

Inferences from System Reliability:

- The overall system reliability has improved with the modifications made to the basic events. However, the improvement in reliability is minimal and could be further enhanced by implementing more significant changes to the DFT.
- With the blue agent having access to additional resources, it wins more games, resulting in a higher reliability percentage.
- Although modifications to the DFT enhance system fault tolerance, the allocation of resources among agents plays a critical role and can either improve or hinder their performance.

7 Conclusion

To conclude, this research project on Analysis of Adversarial Multi-Agent Game Outcomes of Dynamic Fault Tree Games explored the use of DFTs integrated with Adversarial Multi-Agent Reinforcement Learning concept to improve system reliability and fault tolerance. The adversarial agents, red agent for fault injection and blue agent for fault repair, were trained using the DDQN algorithm. Through repeated iterations, agents developed strategies that provided critical insights into the vulnerabilities in the modeled system with the help of analysis techniques.

Analysis of the initial DFT highlighted critical events using Event Usage, Improvement Potential and Reliability analysis. These analysis illustrated the significance of individual events and their roles within the DFT's overall structure. Based on the findings structural modifications, such as adding redundancy and BE parameter improvements, were applied to develop the higher-dimensional DFT.

Subsequent evaluations demonstrated that the system fault tolerance have improved over the higher-dimensional DFT. Redundancy in critical events increased the difficulty of fault propagation, while adjustments to MTTR, repair cost and failure cost introduced strategic trade-offs that favored system resilience. The win rate of blue agent increased, although modestly, indicating that further enhancements to BEs could yield more reliable system.

The study demonstrates the efficacy of combining machine learning techniques with DFT modeling for proactive risk management. By using adversarial simulations, critical vulnerabilities in a system can be identified and mitigated through targeted structural and parametric adjustments.

8 Future Scope

As depicted in the analysis section, the blue agent strategies were week during testing. Optimizing the blue agent's strategies could provide deeper insights into critical events and agent behaviors. By improving the blue agent's performance, the red agent would be forced to adopt more sophisticated strategies and actions, ultimately strengthening the overall reinforcement learning model. This limitation might stem from the use of the DDQN model. Addressing it could involve either enhancing the DDQN algorithm or implementing the project with alternative models such as Dueling-DDQN (D-DDQN) or PPO, which may offer improved performance.

The project implementation is based on a prototype DFT. Future work could also focus on integrating real-world datasets for validation and extending the methodology to more complex systems to enhance its practical applicability and relevance. Automating the modification of the basic events and the parameters of basic events for continuous outcome analysis would streamline the identification of optimal improvement decisions, especially when analyzing complex systems. In that case, introducing a third agent to evaluate the outcomes and implement modifications in the DFT could be explored as a potential avenue for further refinement and efficiency.

Bibliography

AlMahamid, Fadi and Katarina Grolinger (2021). 'Reinforcement Learning Algorithms: An Overview and Classification'. In: *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–7. DOI: 10.1109/CCECE53047.2021.9569056.

Amrutkar, Kalpesh and Kirtee Kamalja (Apr. 2017). 'An Overview of Various Importance Measures of Reliability System'. In: *International Journal of Mathematical, Engineering and Management Sciences* 2, pp. 150–171. DOI: 10.33889/IJMEMS.2017.2.3-014.

Basgöze, Daniel et al. (Feb. 2022). *BDDs Strike Back: Efficient Analysis of Static and Dynamic Fault Trees*. DOI: 10.48550/arXiv.2202.02829.

Boudali, Hichem, Pepijn Crouzen and Mariëlle Stoelinga (July 2010). 'A Rigorous, Compositional, and Extensible Framework for Dynamic Fault Tree Analysis'. In: *Dependable and Secure Computing, IEEE Transactions on* 7, pp. 128–143. DOI: 10.1109/TDSC.2009.45.

Buchholz, Peter and Andreas Blume (2022). 'Dynamic Fault Trees with Correlated Failure Times - Modeling and Efficient Analysis -'. In: *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*, pp. 201–212. DOI: 10.1109/SRDS55811.2022.00027.

Catelani, Marcantonio, Lorenzo Ciani and Matteo Venzi (Sept. 2016). 'Component Reliability Importance assessment on complex systems using Credible Improvement Potential'. In: *Microelectronics Reliability* 64. DOI: 10.1016/j.microrel.2016.07.055.

Espiritu, Jose, David Coit and Upyukt Prakash (Apr. 2007). 'Component criticality importance measures for the power industry'. In: *Electric Power Systems Research* 77, pp. 407–420. DOI: 10.1016/j.epsr.2006.04.003.

Hafiz, A. M. (2023). 'A Survey of Deep Q-Networks used for Reinforcement Learning: State of the Art'. In: *Intelligent Communication Technologies and Virtual Mobile Networks*. Ed. by G. Rajakumar et al. Singapore: Springer Nature Singapore, pp. 393–402. ISBN: 978-981-19-1844-5.

Hafiz, Abdul Mueed and Ghulam Mohiuddin Bhat (2020). 'Deep Q-Network Based Multi-agent Reinforcement Learning with Binary Action Agents'. In: *CoRR* abs/2008.04109. arXiv: 2008.04109. URL: https://arxiv.org/abs/2008.04109.

Huang, Yanhua (2020). 'Deep Q-Networks'. In: *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Ed. by Hao Dong, Zihan Ding and Shanghang Zhang. Singapore: Springer Singapore, pp. 135–160. ISBN: 978-981-15-4095-0. DOI: 10.1007/978-981-15-4095-0_4. URL: https://doi.org/10.1007/978-981-15-4095-0_4.

Jiang, Ke et al. (2019). 'Implementation of a multi-agent environmental regulation strategy under Chinese fiscal decentralization: An evolutionary game theoretical approach'. In: *Journal of Cleaner Production* 214, pp. 902–915. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2018.12.252. URL: https://www.sciencedirect.com/science/article/pii/S095965261833974X.

Le, Bang Giang and Viet Cuong Ta (2024). *Toward Finding Strong Pareto Optimal Policies in Multi-Agent Reinforcement Learning*. arXiv: 2410.19372 [cs.LG]. URL: https://arxiv.org/abs/2410.19372.

Li, Tao, Yuhan Zhao and Quanyan Zhu (2022). 'The role of information structures in game-theoretic multi-agent learning'. In: *Annual Reviews in Control* 53, pp. 296–314. ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2022.03.003. URL: https://www.sciencedirect.com/science/article/pii/S1367578822000086.

Li, Tianxu et al. (2022). 'Applications of Multi-Agent Reinforcement Learning in Future Internet: A Comprehensive Survey'. In: *IEEE Communications Surveys Tutorials* 24.2, pp. 1240–1279. DOI: 10.1109/COMST.2022.3160697.

Liu, Yan, Yue Wu and Zbigniew Kalbarczyk (2017). 'Smart Maintenance via Dynamic Fault Tree Analysis: A Case Study on Singapore MRT System'. In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 511–518. DOI: 10.1109/DSN.2017.50.

Madni, Azad and Michael Sievers (May 2018). 'Model-based systems engineering: Motivation, current status, and research opportunities'. In: *Systems Engineering* 21. DOI: 10.1002/sys.21438.

Merrick, Kathryn E. et al. (2016). 'A Survey of Game Theoretic Approaches to Modelling Decision-Making in Information Warfare Scenarios'. In: *Future Internet* 8, p. 34. URL: https://api.semanticscholar.org/CorpusID:9107060.

Mo, Yuchang (2014). 'A Multiple-Valued Decision-Diagram-Based Approach to Solve Dynamic Fault Trees'. In: *IEEE Transactions on Reliability* 63.1, pp. 81–93. DOI: 10.1109/TR.2014.2299674.

Moradi, Mehrdad, Bert Van Acker and J. Denil (2023). 'Failure Identification Using Model-Implemented Fault Injection with Domain Knowledge-Guided Reinforcement Learning'. In: *Sensors (Basel, Switzerland)* 23. URL: https://api.semanticscholar.org/CorpusID:256932782.

Nowe, Ann, Peter Vrancx and Yann-Michaël De Hauwere (Jan. 2012). 'Game Theory and Multi-agent Reinforcement Learning'. In: p. 30. ISBN: 978-3-642-27645-3. DOI: 10.1007/978-3-642-27645-3_14.

Paté-Cornell, M. Elisabeth (1984). 'Fault Trees vs. Event Trees in Reliability Analysis'. In: *Risk Analysis* 4.3, pp. 177–186. DOI: https://doi.org/10.1111/j.1539-6924.1984.tb00137.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1539-6924.1984.tb00137.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1539-6924.1984.tb00137.x.

Ramos, Ana Luísa, José Ferreira and Jaume Barcelo (Jan. 2012). 'Model-Based Systems Engineering: An Emerging Approach for Modern Systems'. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 42, pp. 101–111. DOI: 10.1109/TSMCC.2011.2106495.

Rizk, Yara, Mariette Awad and E. Tunstel (May 2018). 'Decision Making in Multi-Agent Systems: A Survey'. In: *IEEE Transactions on Cognitive and Developmental Systems* PP, pp. 1–1. DOI: 10.1109/TCDS.2018.2840971.

Roy, Sankardas et al. (2010). 'A Survey of Game Theory as Applied to Network Security'. In: *2010 43rd Hawaii International Conference on System Sciences*, pp. 1–10. DOI: 10.1109/HICSS.2010.35.

Ruijters, Enno and Mariëlle Stoelinga (May 2015). 'Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools'. In: *Computer Science Review* 15-16. DOI: 10.1016/j.cosrev.2015.03.001.

Sinnamon, R.M. and J.D. Andrews (1997). 'New approaches to evaluating fault trees'. In: *Reliability Engineering System Safety* 58.2. ESREL '95, pp. 89–96. ISSN: 0951-8320. DOI: https://doi.org/10.1016/S0951-8320(96)00036-1. URL: https://www.sciencedirect.com/science/article/pii/S0951832096000361.

Tuyls, Karl, Pieter Hoen and Bram Vanschoenwinkel (Jan. 2006). 'An Evolutionary Dynamical Analysis of Multi-Agent Learning in Iterated Games'. In: *Autonomous Agents and Multi-Agent Systems* 12, pp. 115–153. DOI: 10.1007/s10458-005-3783-9.

Tuyls, Karl and Simon Parsons (2007). 'What evolutionary game theory tells us about multiagent learning'. In: *Artificial Intelligence* 171.7. Foundations of Multi-Agent Learning, pp. 406–416. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2007.01.004. URL: https://www.sciencedirect.com/science/article/pii/S0004370207000082.

Wang, Jianrui et al. (May 2022). 'Cooperative and Competitive Multi-Agent Systems: From Optimization to Games'. In: *IEEE/CAA Journal of Automatica Sinica* 9, pp. 763–783. DOI: 10.1109/JAS.2022.105506.

Yevkin, Olexandr (2011). 'An improved modular approach for dynamic fault tree analysis'. In: *2011 Proceedings - Annual Reliability and Maintainability Symposium*, pp. 1–5. DOI: 10.1109/RAMS.2011.5754437.

Zhang, Kaiqing, Zhuoran Yang and Tamer Başar (2021). 'Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms'. In: *Handbook of Reinforcement Learning and Control*. Ed. by Kyriakos G. Vamvoudakis et al. Cham: Springer International Publishing, pp. 321–384. ISBN: 978-3-030-60990-0. DOI: 10.1007/978-3-030-60990-0_12. URL: https://doi.org/10.1007/978-3-030-60990-0_12.

Zheng, J.H. et al. (2023). 'Multi-layer double deep Q network for active distribution network equivalent modeling with internal identification for EV loads'. In: *Applied Soft Computing* 147, p. 110834. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2023.110834. URL: https://www.sciencedirect.com/science/article/pii/S1568494623008529.

Appendix

A Pseudo Code for Calculations

A.1 Calculate Intermediate and Top Event Failure Rates

PS Failure Rate Calculation:
Retrieve the failure rates for events "A" and "B" from the calculated BE failure rates
$ps\_failure\_rate \leftarrow failure\_rate\_A \times failure\_rate\_B$
Store the calculated failure rate accordingly

C1 and C2 Failure Rate Calculation (OR Gates):
$failure\_rate\_C1 \leftarrow failure\_rate\_C + failure\_rate\_D + failure\_rate\_PS$
$-(failure\_rate\_C \times failure\_rate\_D)$
$-(failure\_rate\_C \times failure\_rate\_PS)$
$-(failure\_rate\_D \times failure\_rate\_PS)$
$+(failure\_rate\_C \times failure\_rate\_D \times failure\_rate\_PS)$
Store the calculated failure rate accordingly

$failure\_rate\_C2 \leftarrow failure\_rate\_K + failure\_rate\_L + failure\_rate\_PS$
$-(failure\_rate\_K \times failure\_rate\_L)$
$-(failure\_rate\_K \times failure\_rate\_PS)$
$-(failure\_rate\_L \times failure\_rate\_PS)$
$+(failure\_rate\_K \times failure\_rate\_L \times failure\_rate\_PS)$
Store the calculated failure rate accordingly

M1, M2, M3 Failure Rate Calculation:
$intermediate\_events \leftarrow \{M1, M2, M3\}$
for each event in $intermediate\_events$ **do**
    Retrieve child elements of the event from the gate structure
    Extract the failure rates for child events, defaulting to 0 if not found
    if child failure rates exist **then**
        $failure\_rate \leftarrow child\_failure\_rates[0] + child\_failure\_rates[1]$
        $-(child\_failure\_rates[0] \times child\_failure\_rates[1])$
        Store the calculated failure rate for the event
    end if
end for

CSP1 and CSP2 Failure Rate Calculation:
$csp1\_failure\_rate \leftarrow failure\_rate\_M1 \times (failure\_rate\_M3 + (1 - failure\_rate\_M3) \times failure\_rate\_M2)$
Store the calculated failure rate accordingly
$csp2\_failure\_rate \leftarrow failure\_rate\_M2 \times (failure\_rate\_M3 + (1 - failure\_rate\_M3) \times failure\_rate\_M1)$
Store the calculated failure rate accordingly

G2 and G3 Failure Rate Calculation:
for each event in $\{G2, G3\}$ **do**
    $children \leftarrow gates[event][children]$
    $child\_failure\_rates \leftarrow [gate\_failure\_rates[child]$ for each child]
    if child failure rates exist **then**
        $failure\_rate \leftarrow child\_failure\_rates[0] + child\_failure\_rates[1]$
        $-(child\_failure\_rates[0] \times child\_failure\_rates[1])$
        Store $failure\_rate$ in $gate\_failure\_rates[event]$
    end if
end for

G1 (Top Event) Failure Rate Calculation:
$g1\_failure\_rate \leftarrow failure\_rate\_G2 \times failure\_rate\_G3$
Store the calculated failure rate for the top event

A.2 Calculate Improvement Potential for Basic Events

Original Top Event Failure Rate:
$original\_top\_failure\_rate \leftarrow$ Retrieve Top Event failure rate from $reliability\_analysis$ conducted initially
$original\_failure\_rates \leftarrow$ Make a copy of the current failure rates for all basic events to preserve the original state for later use :$BE\_failure\_rates.copy()$
For Each Basic Event:
for each event in $original\_failure\_rates.keys()$ **do**
    Temporarily set the failure rate of the event considered to 0
    Recalculate the intermediate and top event failure rates
    Store new system failure rate $\leftarrow new\_top\_failure\_rate$
    $improvement\_potential\_value \leftarrow original\_top\_failure\_rate - new\_top\_failure\_rate$
    Store $improvement\_potentials[event] \leftarrow improvement\_potential\_value$
end for

B Initial RL Model Results

B.1 Model I

The event usage during training by the respective agents for the initial model is depicted in the below plots:

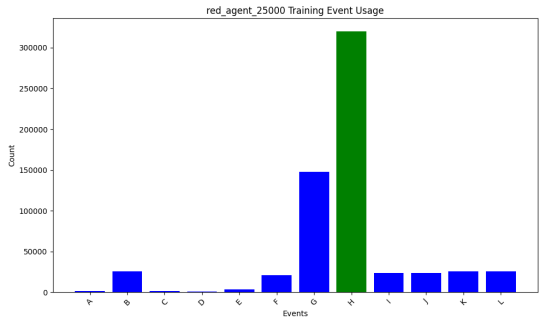


Figure 29: Red Agent Event Usage

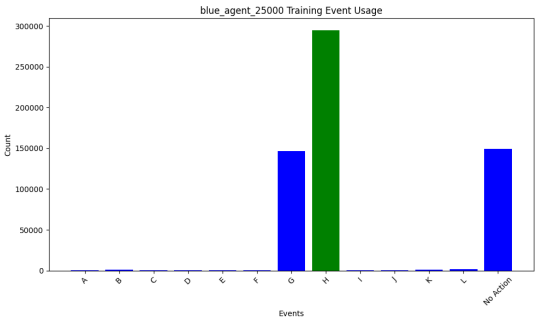


Figure 30: Blue Agent Event Usage

The loss curves for this model is shown below:

Figure 31: Red Agent Loss Curve

Figure 32: Blue Agent Loss Curve

B.2 Model II

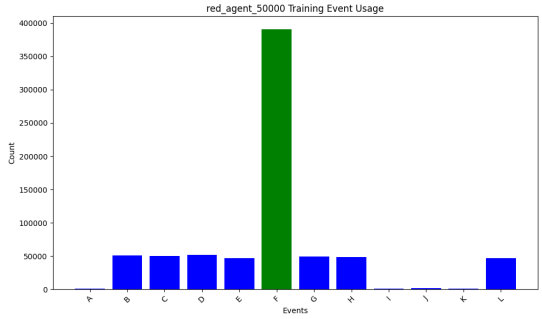


Figure 33: Red Agent Event Usage

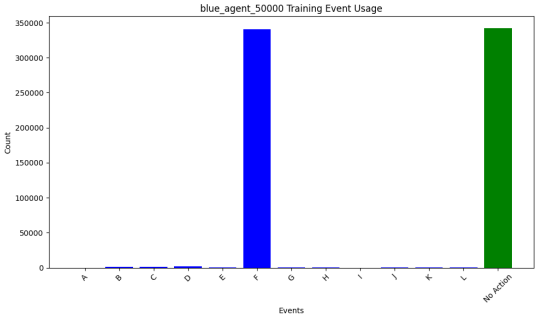


Figure 34: Blue Agent Event Usage

The loss curves for this model is shown below:

Figure 35: Red Agent Loss Curve

Figure 36: Blue Agent Loss Curve

Inference from B.1 and B.2

- The model's performance is suboptimal, and the strategies are weak from a DFT system perspective.
- The red agent chooses one or two events repeatedly, as depicted in the event usage plots. It might be selecting an event multiple times to exhaust the blue agent's resources, then suddenly switching to a different action at the end of the game to cause a system failure.
- While this could be an effective strategy from the red agent's perspective, it is not ideal from the DFT perspective. The focus in DFT is to identify critical events rather than merely causing a system failure through random actions. Moreover, critical events are determined based on sequences of event failures rather than isolated event failures.
- It can also be assumed that the parameters of the events (mean time to repair (MTTR), failure cost, and repair cost) are not being considered by the red agent, as it predominantly injects faults into one or two events. This conclusion is supported by the observation that there are more vulnerable events than those currently being targeted.

Because of these reasons, the outcomes could not be accurately analyzed and would not serve as a reliable source for evaluation. Therefore, the RL model was improved. Some of the improvements made to the RL model are depicted below:

- The Mean Squared Error (MSE) loss was replaced with Huber loss to handle outliers more effectively.
- Gradient clipping was applied during the learning process to stabilize updates.

- Dropout with weight decay was introduced to prevent overfitting.
- A new layer of neurons with batch normalization was added to enhance the model's learning capacity.
- ReLU activations were replaced with Leaky ReLU to address the issue of dead neurons and improve gradient flow, resulting in more stable and efficient training.
- The reward structure was revised to improve the performance of the blue agent: penalties for losing the game were increased, and rewards for winning were significantly boosted.

B.3 Model III

The event usage during training and the loss curves of the respective agents for an improved model are depicted in the below plots:

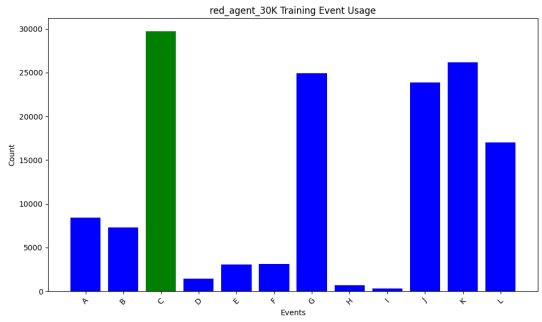


Figure 37: Red Agent Event Usage

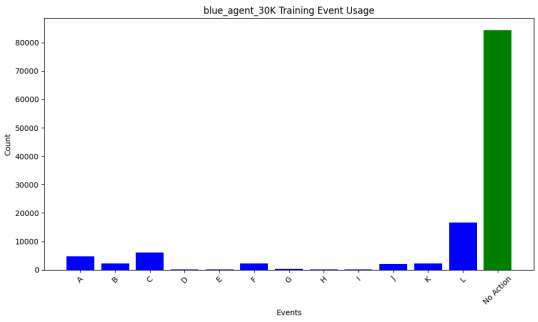


Figure 38: Blue Agent Event Usage

The loss curves for this model is shown below:

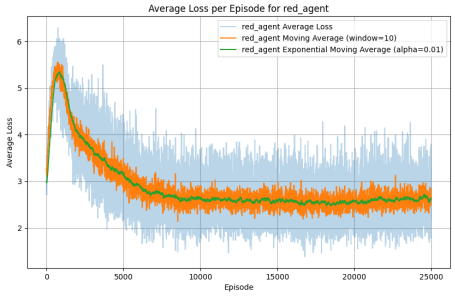


Figure 39: Red Agent Loss Curve

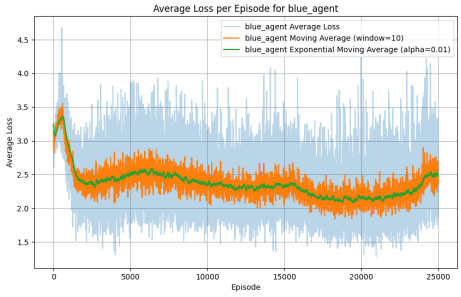


Figure 40: Blue Agent Loss Curve

Note: The model described in B.3 was slightly modified once again to evaluate whether the blue agent's performance could be improved. A training process consisting of 100,000 steps was conducted, and the model was saved for use in the analysis phase of this project.

The loss curves for the model used in this project over 100,000 training steps is shown below:

Using Lower Dimensional DFT

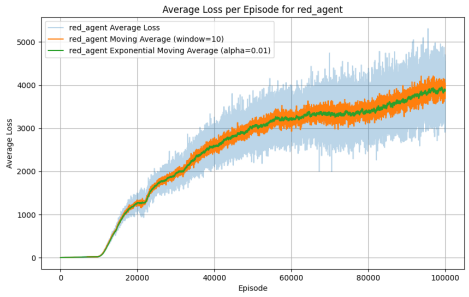


Figure 41: Red Agent Loss Curve

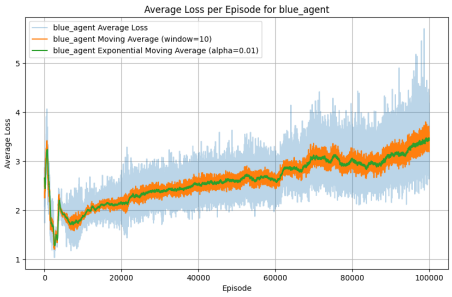


Figure 42: Blue Agent Loss Curve

Using Higher Dimensional DFT

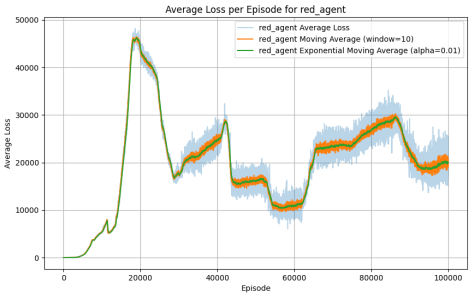


Figure 43: Red Agent Loss Curve

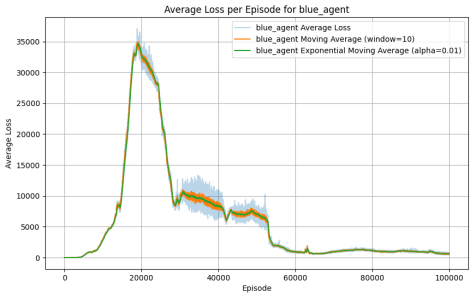


Figure 44: Blue Agent Loss Curve

Declaration of Compliance

I hereby declare to have written this work independently and to have respected in its preparation the relevant provisions, in particular those corresponding to the copyright protection of external materials. Whenever external materials (such as images, drawings, text passages) are used in this work, I declare that these materials are referenced accordingly (e.g. quote, source) and, whenever necessary, consent from the author to use such materials in my work has been obtained.

Date and Signature: 17.12.2024