

Technical University of Munich

TUM School of Engineering and Design

Chair of Computational Modeling and Simulation

Automatic BIM Conflict Resolution Using a Reinforcement Learning Approach

Master Thesis

For the Master of Science Program Resource Efficient and Sustainable
Building

Author: Yuye Jiang

Registration number:

Supervisor: Prof. Dr.-Ing. André Borrmann

Dr. Stavros Nousias

M.Sc. Changyu Du

Date of Issue: 29. February 2024

Date of Submission: 29. August 2024

Abstract

The advent of Building Information Modeling (BIM) has fundamentally altered the way how building projects are approached. Nevertheless, it is still the case that conflicts during the collaboration process are an unavoidable consequence, resulting in inefficiencies. The widespread adoption of BIM model checker has significantly contributed to the detection of conflicts. However, the automation of conflict resolution remains a nascent endeavor, and the reliance on manual communication among designers can be a significant time investment. To address this shortcoming, this research presents a framework for training a Reinforcement Learning (RL) agent, employing the Proximal Policy Optimization (PPO) algorithm, in the integrated real BIM environment to automate the conflict resolution process, with a particular focus on geometric conflicts. Three experiments, each focusing on a different type of conflict, were conducted to investigate the feasibility of the proposed framework. The results were analyzed, and the limitations were discussed.

Zusammenfassung

Die Einführung von Building Information Modelling (BIM) hat zu einer grundlegenden Veränderung der Art und Weise geführt, wie Bauprojekte angegangen werden. Dennoch können Konflikte während der Zusammenarbeit nicht gänzlich ausgeschlossen werden, was zu Ineffizienzen führt. Die weit verbreitete Anwendung von BIM Modell-Checker hat maßgeblich zur Identifikation von Konflikten beigetragen. Die Automatisierung der Konfliktlösung befindet sich jedoch noch in den Anfängen und die Abhängigkeit von der manuellen Kommunikation zwischen den Planern kann einen erheblichen Zeitaufwand bedeuten. Um dieses Defizit zu beheben, wird in dieser Forschungsarbeit ein Rahmen für das Training eines Reinforcement Learning (RL)-Agenten vorgestellt, der den Algorithmus Proximal Policy Optimization (PPO) in der integrierten realen BIM Umgebung einsetzt, um den Konfliktlösungsprozess zu automatisieren, wobei ein besonderer Schwerpunkt auf geometrischen Konflikten liegt. Zur Evaluierung der Machbarkeit des vorgeschlagenen Rahmens wurden drei Experimente durchgeführt, welche sich jeweils auf eine spezifische Art von Konflikten ausgerichtet. Die Ergebnisse wurden umfassend analysiert und die Limitationen der Experimente und des Rahmens diskutiert.

List of Contents

List of Figures	VI	
List of Tables	VII	
List of Abbreviations	VIII	
1	Introduction	1
1.1	Motivation	1
1.2	Research Objectives.....	3
1.3	Reading Guide.....	3
2	Related Works	5
2.1	BIM-Based Model Checking	5
2.2	Automatic Conflict Resolution in BIM model	7
2.3	The Application of RL in General BIM.....	8
2.4	The Research Gap.....	10
3	Theoretical Basis	12
3.1	Industry Foundation Classes File.....	12
3.2	Application Programming Interface	14
3.3	Reinforcement Learning Algorithm.....	16
3.3.1	Reinforcement Learning Fundamentals	16
3.3.2	Classification of RL Algorithms	19
3.3.3	Proximal Policy Optimization	22
4	Methodology	26
4.1	State Module.....	27
4.2	Action Module	30
4.3	Reward Module.....	31
4.4	PPO-based Conflict Resolution Algorithm	33
5	Implementation Details	35
5.1	Model Checker.....	36

5.1.1	Solibri Rulesets	36
5.1.2	Solibri REST API.....	38
5.1.3	Solibri Java API.....	39
5.2	IfcOpenShell	40
5.3	Stable Baselines3	41
6	The Experiments and Evaluation	43
6.1	The Experimental Environment and Configurations.....	43
6.2	The Toilet-Wall Conflict.....	44
6.2.1	The Experiments.....	45
6.2.2	Results and Analysis.....	47
6.3	The Column-Window Conflict	48
6.3.1	The Experiments.....	48
6.3.2	Results and Analysis.....	50
6.4	The Air Terminal-Door Conflict.....	51
6.4.1	The Experiments.....	51
6.4.2	Results and analysis	54
7	Conclusions and Future Works	55
7.1	Conclusions	55
7.2	Limitations.....	56
7.3	Future Works	58
	References	60
	Appendix A	67

List of Figures

Figure 3.1: An Example Diagram of IfcWindow	13
Figure 3.2: The Schematic Diagram of API	14
Figure 3.3: The Schematic Diagram of RL.....	17
Figure 3.4: The clipping of PPO's surrogate objective function	24
Figure 4.1: The framework of this study	26
Figure 5.1: Implementation of the RL system	35
Figure 5.2: An example of the predefined rulesets in Solibri.....	37
Figure 5.3: An example of the extracted checking results	39
Figure 6.1: The IFC model used for training	44
Figure 6.2: The initial placement regions of the toilet seat.....	46
Figure 6.3: The illustration of the agent movement in the first training episode	47
Figure 6.4: The training evaluation for the toilet-wall conflict	47
Figure 6.5: The initial placement area of the column	49
Figure 6.6: The training evaluation for the column-window conflict	51
Figure 6.7: The illustration of the air terminal-door conflict	52
Figure 6.8: The illustration of the agent for the air terminal-door conflict.....	53
Figure 6.9: The training evaluation for the air terminal-door conflict.....	54

List of Tables

Table 4-1: Summary of the observation parameters..... 28

Table 4-2: Summary of the action..... 30

Table 5-1: The commands for launching Solibri using REST API..... 38

List of Abbreviations

BIM	Building Information Modeling
AEC	Architectural, Engineering, and Construction
RL	Reinforcement Learning
MEP	Mechanical, Electrical, Plumbing
IFC	Industry Foundation Classes
BCF	BIM Collaboration Format
AI	Artificial Intelligence
SL	Supervised Learning
API	Application Programming Interface
PPO	Proximal Policy Optimization
ML	Machine Learning
DRL	Deep Reinforcement Learning
RC	Reinforced Concrete
MARL	Multi-Agent Reinforcement Learning
GAN	Generative Adversarial Network
DDPG	Deep Deterministic Policy Gradient
MVD	Model View Definition
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer

MDP	Markov Decision Process
DL	Deep Learning
SARSA	State-Action-Reward-State-Action
TRPO	Trust Region Policy Optimization
MPC	Model Predictive Control
KL	Kullback-Leibler
UUID	Universally Unique Identifier
GUID	Globally Unique Identifier
CSV	Comma-Separated Values
SB3	StableBaselines3
MLP	Multilayer Perceptron
ReLU	The Rectified Linear Unit

1 Introduction

Building Information Modeling (BIM) offers a novel approach to design, construction, and facility management, in which a digital representation of the building product and process is used to facilitate the exchange and interoperability of information (Sacks et al., 2018). BIM has revolutionized the workflow for building projects in the Architectural, Engineering, and Construction (AEC) industry. However, the process of BIM modeling often necessitates the collaboration of designers from different disciplines, which can give rise to conflicts. These conflicts are recognized as critical causes of deficiency and low performance in building projects (Charehzehi et al., 2017). Despite the availability of many BIM model checker tools that can automatically detect conflicts, the current state of conflict resolution remains a slow manual process that requires continuous communication between responsible experts. In response to this challenge, our research explores the possibilities to apply Reinforcement Learning (RL) techniques to automate the BIM conflict resolution process.

1.1 Motivation

Collaborative teamwork is an essential aspect of a modern BIM project, which can be in two forms as, synchronous and asynchronous collaboration (Isikdag & Underwood, 2010). Synchronous collaborative teamwork denotes a situation in which all members of a team are working on the same model at the same time and exchanging their expert knowledge simultaneously (Scherer, 2007). This approach to working is typically employed within the confines of a single design office, utilizing the specialized BIM authoring software. Asynchronous collaboration means that each involved discipline team, including architectural engineering, structural engineering, and mechanical, electrical, plumbing (MEP) engineering, creating their own model and conducting their design process independently and at disparate locations. The project leader or BIM coordinator of each discipline communicates and exchanges information at frequent intervals and merges these sub-models into a federated model. When BIM subsets are integrated into a single BIM using appropriate software platforms, the design parameters of different disciplines are most likely to conflict with each other, given the high complexity of building models (Chen & Hou, 2014).

In order to achieve a conflict-free integrated BIM model, it is necessary to detect the existing conflicts. The same BIM authoring software is typically utilized by all team members within a single company. Accordingly, the detection and sharing of issues can be accomplished via the commercial cloud BIM platform provided by the corresponding software vendor, such as BIM 360 from Autodesk, BIMcloud from Graphisoft, and BIMPLUS from Nemetschek. In contrast, this exchanged workflow between companies that use different BIM authoring tools, often based on production of Industry Foundation Classes (IFC) files and BIM Collaboration Format (BCF) files (Kubicki et al., 2019). The IFC file is a vendor-neutral, industry-specific data model schema that enables the sharing of information throughout the project lifecycle, while the BCF file stores text, images, and component information about the conflicts, allowing different BIM applications to communicate model-based issues (buildingSMART, 2024). Details about the IFC file will be explained further in section 3.1. Once the detection of conflicts is complete, the engineers responsible for the issues need to resolve the conflicts manually. This process frequently involves extensive collaboration and discussion with other disciplines until the integrated model is either conflict-free or at least acceptable without significant clashes.

In regard to the aforementioned conflict resolution workflow, the detection process can be accomplished with a very high degree of automation. Most BIM cloud platforms can perform at least simple collision detection and categorize these issues according to the disciplines to which the conflicted element belongs. Other widely adopted BIM-based model checkers, such as Solibri, contribute significantly to conflict detection. In addition to geometrical collisions between elements, they can also check the accessibility of space, fire protection requirements, component properties, and numerous other building standards and regulations.

Nevertheless, even with the utilization of model checker tools to automate conflict detection, conflict resolution in BIM projects remains a slow and manual process (Harode et al., 2024). The automation of conflict resolution is currently very limited, and the intensive communication process among designers tends to be a significant time investment. In response to this challenge, research has already begun to explore the potential of utilizing various Artificial Intelligence (AI) technologies to automate the process. For instance, Supervised Learning (SL) techniques can be employed to collect existing clash-resolution data and experts' opinions with the objective of training a model to resolve clashes automatically (Hsu et al., 2020). This type of research has

yielded promising results. However, the SL learns from a set of provided labeled examples, which presents a critical disadvantage in the lack of sufficient data within the AEC sector.

In distinction to the SL, the RL is applicable when examples of desired behavior are not available but where it is possible to evaluate examples of behavior according to some performance criterion (Si et al., 2009). Consequently, our research aims to explore the application of RL as an approach to automate the conflict resolution process, thereby reducing reliance on manual labor and improving the efficiency of this pivotal aspect of BIM development.

1.2 Research Objectives

The principal objective of this thesis is to:

- Set up a RL environment that integrates the real-world BIM model and rule-based model checker to train an agent to autonomously resolve geometric conflicts.

The aim of this approach is to achieve a high degree of automation, requiring no input of data or specific prior knowledge, with the intention of reducing the laborious manual work involved in the conventional conflict resolution process.

1.3 Reading Guide

This thesis is structured in the following chapters:

- Chapter 2 provides an overview of state-of-the-art research. The studies are classified into three categories based on the research topic. Their limitations are discussed, and the research gaps are summarized.
- Chapter 3 outlines the hierarchical structure of the IFC file, the principle of software Application Programming Interface (API), and the underpinnings of the RL algorithm, with a particular emphasis on the Proximal Policy Optimization (PPO). This information provides a useful theoretical basis for understanding the proposed framework.
- Chapter 4 details the methodology of the research, which is of crucial importance for the thesis. The integration of the RL environment with the model checker using IFC file, and the application of PPO are explained in depth.

-
- Chapter 5 describes the practical implementation of the proposed method, illustrating the translating of the theoretical framework into practice by employing Solibri APIs and diverse Python libraries.
 - Chapter 6 demonstrates three distinct use cases and the configuration of the experimental setting for each. The training results of the RL algorithm are presented and analyzed individually.
 - Chapter 7 addresses the research questions, discusses the research findings, and summarizes the contributions of this thesis. The limitations of the study and the future possibility of the proposed framework are also provided.

2 Related Works

In this section, the state-of-the-art studies related to this research are presented. The findings of various studies on model checking can offer insights into the BIM conflicts. The application of different algorithms and methods to automate the resolution of conflicts can demonstrate the prevailing direction and advancements being made in this field. Furthermore, the research of the employment of RL in the broader BIM and modeling domain can provide valuable references of the practical transformation of a real-world problem into a modeled one. At the conclusion of this section, the characteristics and limitations of these studies will be summarized, thereby identifying the research gap that remains to be explored.

2.1 BIM-Based Model Checking

BIM model conflicts encompass a multitude of different issues. As a prerequisite for solving them, it is important to first comprehend how conflicts in BIM are identified and detected, as well as which parameters are of paramount importance for defining conflicts. By doing so, we can ascertain the most suitable tool for detecting conflicts in our research and concentrate on the most prevalent conflicts that can be resolved using RL, while also acknowledging the diversity of conflicts under study. Consequently, research on model checking itself is also crucial.

With the help of work breakdown structures, using better organized 3D BIM model to detect clash has been proven more efficient and accurate than using 2D drawings (Gijezen et al., 2010). Because of the hierarchy structure present in IFC, in addition to the 3D shape of building objects, clash detection using the IFC standard has been shown to have higher performance (Tizani, 2010). Two of the fastest ways to calculate geometric collisions are presented: (1) comparing the distance between the centers and the sum of the radiuses of two spheres, and (2) comparing axis-aligned bounding boxes. More accurate detection requires the calculation of ray-triangle intersections.

A formalized knowledge representation for spatial conflict coordination was proposed for clash documentation (Wang & Leite, 2016). It comprises four categories:

1. Clash description includes information used to describe the objects, such as system name, coordinate, length, object type etc.
2. Clash context represents two types of information: topological context, such as location, and constraints.
3. Clash evaluation contains clash severity, clash cause and solution
4. Clash management involves the identification and monitoring of the coordination process.

A knowledge base management system was integrated with BIM to improve the automated rule checking specifically for MEP systems (Xie et al., 2022). They identified three modules for the proposed checking system: system integrity checking, component's property checking, and spacing constraints of element checking. While current BIM model checker has facilitated the easy detection of conflicts, a notable proportion of these conflicts are either permissible or tolerable. However, the model checker usually does not adequately reveal this information. To address this shortcoming, a rule-based reasoning system and Machine Learning (ML) classifiers were implemented (Huang & Lin, 2019). This research took structural clashes against the MEP model as a study example and revealed that the two most important factors for determining the severity of clashes are the type of the two clashing components and their placement.

The extant literature underscores the complexity and diversity of model conflicts in BIM, emphasizing the superiority of 3D BIM models and the utility of IFC format. A variety of methodologies have been proposed to enhance the detection and identification of conflicts. In light of the inherent impossibility of resolving all conflicts at once, the act of prioritization assumes considerable importance. Geometrical spatial conflicts are of particular significance and have been the focus of most studies. It is frequently observed in the literature that two elements are in conflict with each other. Even in instances where there are more than two elements involved, they can be effectively reduced to a two-by-two conflict structure. Consequently, our research endeavors have been directed towards the resolution of two-element geometrical conflicts. Some of the key factors, such as element type, coordinates and location, were repeatedly highlighted in different studies, which are very informative for our research, especially for the definition of custom RL environment state.

2.2 Automatic Conflict Resolution in BIM model

With the increasing attention on BIM conflicts, several noteworthy studies have been conducted to the development of automated solutions for BIM-based clash resolution. These efforts have employed various AI technologies, including RL algorithms, to enhance the efficiency and accuracy of conflict resolution processes. The integration of AI into BIM workflows has led to significant advancements, allowing for more sophisticated analysis and decision-making capabilities.

In the early years, researchers proposed an automated approach for identifying and resolving spatial clashes in MEP design (Radke et al., 2009). The proposed resolution involved moving one of the two clashing entities to solve spatial conflicts. This basic approach can be seen in many subsequent studies. However, due to the technological limitations at that time, the approach still required much human intervention and did not utilize AI. An AI solution incorporating knowledge-based ML and heuristic optimizing techniques was developed to address design clashes in BIM model (Hsu et al., 2020). Five experienced constructors were invited to participate extensively by completing a questionnaire. The results were collected, the underlying knowledge pattern was analyzed, and then used as the basis for optimization. The questionnaire and the test of the optimization approaches were all conducted in the same experimental environment, which focused on the MEP systems in the basement of an actual student residence.

Since one conflict can have different solutions, a fuzzy analytic hierarchy process was proposed to prioritize various collected optimal clash resolution methods, and a Navisworks plug-in was designed to automatically display the available resolutions (Hassannejad et al., 2023). The weighting and prioritizing criteria in this research were also obtained through questionnaires and expert consultations. Furthermore, the final decision was still required to be manually selected, and no additional clash detection was conducted subsequent to the application of the resolution. To simplify the clash resolution process, most research primarily focused on moving a single objective, which can be challenging in practice. Therefore, considering it as a multi-objective optimization problem, a second-generation non-dominated sorting genetic algorithm approach was proposed to balance the optimization of multiple objectives (X. Liu et al., 2024).

Harode et.al. have investigated the general application of Supervised Learning (SL) for clash resolution (Harode & Thabet, 2021). In their paper, the limitations of SL were

pointed out, and a methodology combining SL and RL was proposed. The following year, the authors explained this methodology in more detail (Harode et al., 2022). A probabilistic model using an SL algorithm was designed. Information such as element type, Revit category, orientation for both clashing elements, and system priority were collected as input variables while the possible clash resolution decisions (moving elements up, down, left, right, changing the length or slope, and rerouting) were designed as output variables. In the follow-up study, they further explored the common strategy by adopting an artificial neural network prediction model that could suggest possible clash resolution options, achieving an accuracy of 84% at highest (Harode et al., 2024). In conclusion, many researchers have dedicated their efforts to improving automatic BIM-based conflict resolution, employing a diverse array of AI technologies. However, the majority of these approaches require input from existing data and the involvement of experienced experts, and the application scenarios are also very limited. There exists a notable gap in the application of RL. Some researchers have put forth the concept of integrating SL and RL, developing a framework to facilitate the integration. Nevertheless, in their subsequent and more in-depth research, they devoted a significant portion of their efforts to SL, while the research on the RL aspect has remained relatively stagnant.

2.3 The Application of RL in General BIM

The rapid advancement of RL in recent years has demonstrated its potential for solving complex problems across a range of domains. One notable example is the development of AlphaGo (Silver et al., 2016), which outperformed human champions in the strategic game of Go. In this section, we focus on research papers that employ RL to address topics similar to or highly relevant to conflict resolution in BIM, including model merging and reinforcement steel design. These studies illustrate the integrating role of RL within BIM, offering valuable practical examples.

While situated in a different domain, general model merging issues bear significant similarity to the conflicts that arise from model collaboration in the context of BIM. An RL approach to automatically resolve merging conflicts based on quality characteristics, as introduced by language modeling engineers as preferences, was proposed to address model merging issues (Sharbaf et al., 2022). This RL-based resolution algorithm does not require initial training data; however, the quality evaluation still heavily relies on the involved engineers. Yang et al. proposed a Deep Reinforcement Learning (DRL)

method for generating three-dimensional pipeline layouts (Yang et al., 2023). In this study, a highly simplified simulated training environment was constructed, containing several key elements in the space that affect pipeline layout. A series of rules were defined to ensure that the generated space complies with real building standards and that the geometric constraints of the pipelines prevent them from colliding with or passing through the obstacles or walls.

The design of Rebar in Reinforced Concrete (RC) structures is a crucial stage in the structural modeling of BIM projects. Though the employment of BIM technology can assist in the design process, achieving a clash-free design for each joint, particularly with irregularly shaped RC structures and rebar, still demands labor-intensive manual work. A framework targeted at this specific task was presented, combining a Multi-Agent Reinforcement Learning (MARL) system with BIM (J. Liu et al., 2019). Each rebar was regarded as an intelligent RL agent. The rebar design problem was formulated as a three-dimensional path-planning problem. The authors described the proposed RL framework in general terms using a fusion architecture for learning, cognition, and navigation engines. The following year, they extended this work and implemented Q-learning for a more realistic design of real-world RC structures (J. Liu et al., 2020). The state, action, and immediate and delayed rewards for the MARL were designed with consideration of actual constructible constraints and design codes. The jointed columns or beams were designed individually in the transformed grid-based digital environment. Each designed structure was considered an obstacle for the next, ensuring the complete design was clash-free. Comprehensive experiments on three typical beam-column joints in a two-story RC building frame were conducted to evaluate the proposed method. The average success rates and time spent confirmed that the proposed framework is efficient and effective. The same framework was extended to automatically generate clash-free rebar designs in prefabricated concrete wall panels, integrating a Generative Adversarial Network (GAN) to learn from designers' experiences with existing design drawings and generate 2D preliminary rebar designs (P. Liu et al., 2023). Similarly, another RL method was proposed based on the Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al., 2019), with the objective of handling higher-dimensional tasks in continuous action space and designing an RC beam in a cost-effectively manner while considering both flexural and shear reinforcement arrangements (Jeong & Jo, 2021).

It is notable that the utilization of RL has achieved remarkable outcomes in related fields, implying the potential for the practical application of RL in the context of automatic BIM conflict resolution. These approaches to transforming real-world problems, simplifying, and modeling the environment are worthy of emulation. A number of RL algorithms, including Q-learning, DQN, and MARL, have been demonstrated to resolve different problems, thereby showing the extensibility of the RL framework. However, this also underscores the importance of selecting the appropriate algorithm.

2.4 The Research Gap

The results of the literature review demonstrated that the resolution of conflicts in BIM model has attracted considerable attention from researchers in recent years. Many conflict detection methodologies have been proposed, and various tools have been developed to automatically or semi-automatically address conflicts that emerge during the collaboration process. However, despite the progress made in BIM clash resolution, there remains significant gaps in the research, which are reflected in the following aspects:

- Limited exploration of RL: Although a framework combining RL and SL has been proposed, the direct application of RL for general BIM conflict resolution has been scarcely studied. Researchers have focused more on automatic design within BIM, particularly in RC design.
- Demand for data: As observed in numerous studies, researchers have proposed a range of ML algorithms to resolve conflicts, which require a large quantity of training data to achieve satisfactory results (Sutton & Barto, 1998). However, as previously stated, the dearth of adequate datasets in the BIM domain poses a formidable challenge for implementing SL in this field. While the input from several experts may be suffice for specific use cases, this approach is evidently limited by the number and expertise of the experts involved and is difficult to generalize to other scenarios.
- Simplification of the BIM environment: Due to the complexity of real BIM model, most research studies employed significant simplifications, extracting only the essential information to create a simulated model and environment for RL training. However, a BIM model is an integrated whole, with complex interrelations among its objects, making simplifications less than ideal.

-
- Limited use cases: Most research has concentrated on a specific type of direct clash within BIM model, with MEP systems representing the most extensively studied discipline due to their complexity. However, architectural conflicts, structural conflicts, and interdisciplinary conflicts are also prevalent. Furthermore, in addition to clashes, there are numerous other types of geometrical conflicts. As a result, the generalization of the objects studied is limited.

Given the aforementioned findings and current limitations, our research will focus on the practical application of an RL approach for a real IFC model for geometric conflict resolution. No extensive data is required, and the widely used RL algorithm, PPO, will be employed. This algorithm is known for its balance between simplicity and performance. The integration of powerful BIM model checker facilitates the seamless detection of all conflicts within the IFC model throughout the training process, thereby ensuring comprehensive compliance across the entire model. To illustrate the versatility and adaptability of our framework, three use cases will be examined, encompassing a diverse range of geometric conflicts from various disciplines. This study aims to bridge the existing research gaps in the field of automatic BIM conflict resolution.

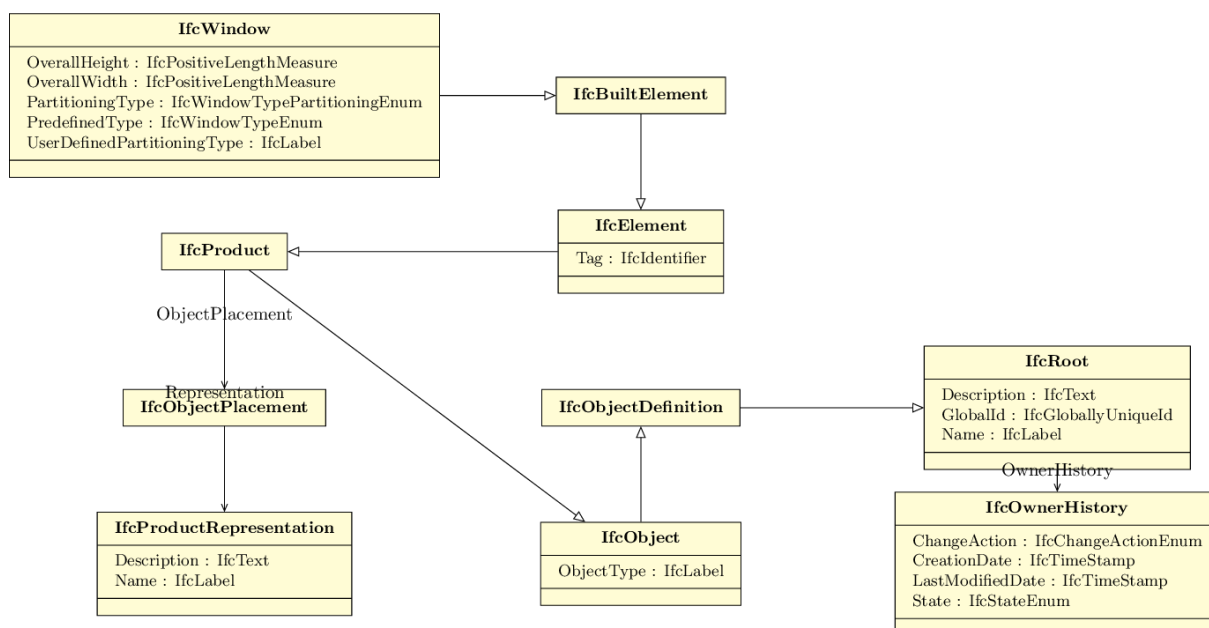
3 Theoretical Basis

This chapter presents several fundamental theoretical concepts relevant to the proposed approach for BIM conflict resolution. It covers the introduction of IFC file format, the principles of API, and an overview of RL with a particular focus on the PPO algorithm. This foundation provides a comprehensive guide for understanding the methodology of this thesis.

3.1 Industry Foundation Classes File

In comparison to existing research, the IFC model was selected as the data representation format for the entire workflow rather than relying on a specific model format of BIM authoring software, to enhance generalizability of our approach. Industry foundation classes is an international standardized digital representation of the built asset developed by buildingSMART. The IFC model has been certified under ISO 16739-1:2018 and is designed to promote vendor-neutral and usable capabilities across a wide range of hardware devices, software platforms, and interfaces for various use cases (buildingSMART technical, 2024). To date, the buildingSMART team has developed the latest official version, IFC 4.3. Furthermore, the majority of BIM authoring tools offer export and import functionalities for IFC files, ensuring the interoperability of IFC across the AEC sectors.

IFC is a complex data model that allows for the representation of both the geometrical and semantic structures of a building model using an object-oriented approach (Borrmann et al., 2018). Each object type, such as walls, windows, and columns, has a hierarchical representation that defines its specialization and generalization relationships. All objects and their relationships together constitute the digital building environment. This structure is crucial for ensuring that the IFC model can be accurately decomposed into simpler parts, which is necessary for downstream applications, such as simulation, analysis, and quantity take-off. To gain a deeper comprehension of this structure, the schema of a conventional building component, the window, is presented as an illustrative example in Figure 3.1:



- Figure 3.1: An Example Diagram of *IfcWindow* (buildingSMART, 2024)

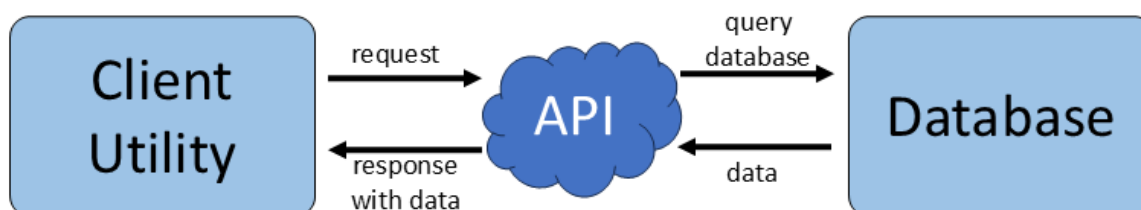
From the instance of *IfcWindow*, the concept of entity inheritance is clearly understood. The hierarchy commences with the specific building element type, *IfcWindow*, and progresses up to the most abstract and root class, *IfcRoot*. Each class possesses a distinct set of attributes and dependencies. This hierarchy follows the semantic relationship chain, yet it also includes interrelationships with geometric representations, such as the classes *IfcObjectPlacement* and *IfcProductRepresentation*, along with their subclasses, which are not detailed in the schema. The pre-defined attributes, some of which are specific and can be inherited by other classes, function to organize information clearly, ensuring correct and straightforward interpretation and extraction by a variety of platforms, including Python. Furthermore, there are classes like *IfcBuildingElementProxy*, which accommodate non-standard building elements, providing similar functionality to any subtype of *IfcBuildingElement* without a predefined meaning.

In addition to the hierarchical structure, an object is also defined by various relationships, such as spatial relationships that link it to a story, building, site, and ultimately the project. The complexity and extensiveness of the IFC model are evident, which is why a Model View Definition (MVD) is introduced to define a subset of the full data for uniform exchange scenarios. For IFC 4, two standard pre-defined MVDs are available: the Reference View and the Design Transfer View. The latter contains more detail for scenarios where continued design and editing are required.

IFC plays a pivotal role in BIM project workflows. It has become the mandatory hand-over format for public building projects in countries like Singapore (Building and Construction Authority, 2013). Given the necessity for multidisciplinary teams to exchange and merge IFC files in the course of their collaboration in BIM, IFC is a suitable choice for our research, which aims to encompass conflicts across all architectural, structural, and MEP disciplines.

3.2 Application Programming Interface

An application programming interface is a set of rules or protocols that enables software applications to communicate with each other, facilitating the exchange of data, features, and functionality (IBM, 2024). Essentially, an API defines the methods and data structures that clients or developers can use to interact with an application, service, database, or component, often without needing to understand its underlying implementation. APIs are fundamental to modern software, enabling efficient and seamless collaboration between different applications and services.



• Figure 3.2: The Schematic Diagram of API

APIs work by facilitating data exchange between applications, systems, and devices through a request and response cycle. As illustrated in Figure 3.2, a request is sent to the API, which then retrieves the data and returns it to the user (Postman, 2024). APIs can be classified in various ways. One such classification is based on access levels, which distinguishes between three types: (1) private APIs, (2) public APIs, and (3) partner APIs. Another classification is based on use cases, which identifies four types: (1) data APIs, (2) operating system APIs, (3) remote APIs, and (4) web APIs. This paper will focus on public operating system APIs and public web APIs.

Operating system APIs, also referred to as local APIs, delineate how applications utilize the services and resources of the operating system. Examples include Java APIs and Python APIs, which provide a set of functions for interacting with the operating system, local file systems, networks, graphical user interfaces, and databases. These

APIs require an understanding of the underlying logic of the software, assisting developers in the creation, management, and control of applications at the local level in accordance with their custom needs. In contrast, web APIs facilitate interaction over the internet, often using Hypertext Transfer Protocol (HTTP) as the communication protocol. They are commonly used for services such as data retrieval, remote procedure calls, and interaction with third-party platforms. One popular architectural style for web APIs is the REpresentational State Transfer (REST) API.

A REST API, also known as a RESTful API, is based on a stateless, client-server communication model where the client interacts with resources on the server using standard HTTP methods. Each resource is represented by a unique uniform resource identifier, and four typical methods correspond to different types of operations are:

- GET
- POST
- PUT
- DELETE

APIs are highly versatile and can be utilized to support a multitude of use cases, including the integration of internal systems, the enhancement of functionality, and the automation of repetitive tasks. This versatility makes them an appropriate instrument for our research. Given that a building is an integrated complex with intricate interrelationships between components, the relocation of one object to resolve a conflict may result in violating spatial constraints of other objects, potentially causing new conflicts. Abstracted training environments are ill-equipped to respond to this problem, even studded up with custom checking features. This is because such customized detection functions are typically rudimentary, and the vast majority of other elements within the model are not abstracted into the training environment. Consequently, the optimal solution is to employ a comprehensive BIM model and a robust model checker to oversee the training process. Leveraging a well-established BIM model checker and its public APIs for developers can effectively address this challenge.

3.3 Reinforcement Learning Algorithm

Machine learning is a computational method that uses experience to improve performance or make accurate predictions (Mohri et al., 2018). Based on how and what experience is transferred to the machine, machine learning can be divided into three categories: (1) Supervised Learning, (2) Unsupervised Learning, and (3) Reinforcement Learning. In our research, we focus on Reinforcement Learning.

3.3.1 Reinforcement Learning Fundamentals

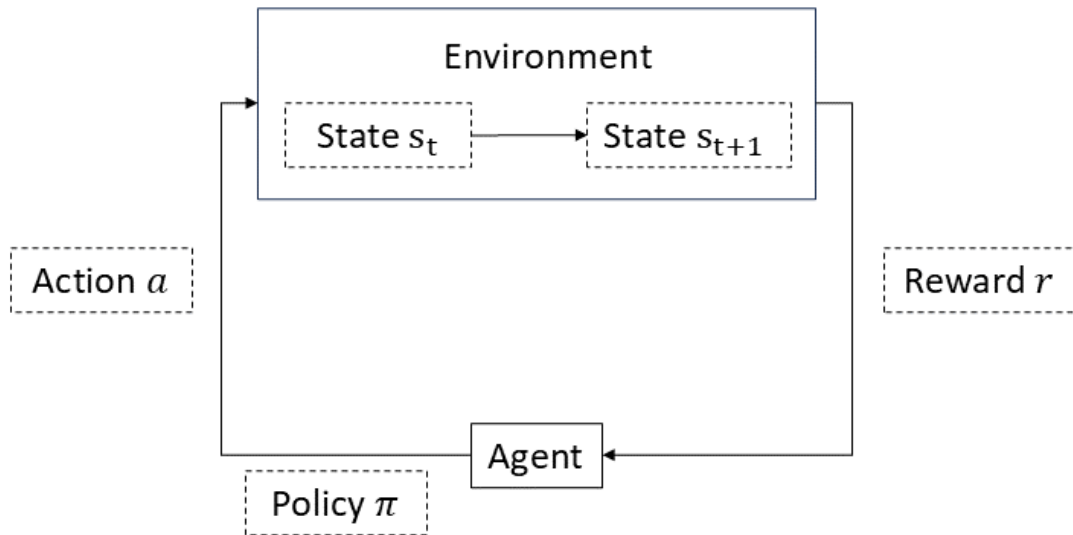
In the field of reinforcement learning, the interaction between the agent and the environment is modeled using the mathematical framework known as the Markov Decision Process (MDP). Unlike the other two types of ML, RL does not require initial training data, whether labeled or not. Instead, the RL model explores its environment freely, gathering knowledge based on its successes and failures. The RL model then applies this self-collected knowledge to adjust its behavior, thereby enhancing its accuracy in prediction and interaction with its environment.

A RL model consists of two components:

1. Agent: the entity that observes the environment, inferences, makes decisions and takes action.
2. Environment: the external system with which the agent interacts.

The learning process of an RL agent is analogous to human learning. For each successful interaction with the environment, the agent receives a numerical reward to encourage similar future interactions. During the interaction between the agent and the environment, four important terminologies are involved:

1. State (s): represents the information contained in the environment at a specific point in time.
2. Action (a): refers to the interactions or decisions the agent makes with the environment.
3. Policy (π): the strategy that the agent employs to determine its actions based on the current state.
4. Reward (r): the numerical value the agent received from the environment after taking an action.



- [Figure 3.3: The Schematic Diagram of RL](#)

The schematic diagram in the Figure 3.3 illustrates the training process of RL. The agent observes the current state (s_t) of the environment at a given time t and takes an action (a_t) that is determined by the policy (π). The policy is a mapping that represents the probability of taking a specific action given the state (s_t), denoted as $\pi(a | s_t)$. As a consequence of the action, the state of the environment transitions from s_t to s_{t+1} . The agent also receives a feedback signal, namely the reward (r_t) based on the success or failure of the action. This one iteration of the agent-environment interaction is defined as a step. The iterative loop continues until the environment reaches its terminated state by successfully performing a sequence of actions or reaches its truncated state by triggering a certain condition. The sequence of steps that starts with the initial state and end with the end state is defined as an episode.

During the training phase, the agent refines the policy and learns how to respond to states with appropriate actions that maximize the total reward. The reward received by the agent at each step is an immediate reward, designated as r_t , while the goal of RL is to maximize long-term cumulative rewards. Therefore, in addition to the current immediate reward, it is necessary to consider the potential rewards that may be obtained in the future. The total accumulated future reward that an agent receives over time, starting from a specific time step t , is referred to as the return G_t :

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \gamma^3 \cdot r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k}$$

The discount factors γ is introduced into the formula to represent the preference for immediate rewards over future rewards. The return is a crucial calculation concept in RL, as it serves as the basis for evaluating the effectiveness of policies, with the objective being to maximize the expected return. In particular, the State-Value function and the Action-Value function of a policy can be defined to measure the expectation of the return G_t from a specific state under a certain policy and the expectation of the return G_t from a specific state after executing a specific action.

The State-Value function:

$$V_{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

Here

$V_{\pi}(s)$: The expectation of the return that can be achieved by executing the policy π in the current state s

π : The current policy being followed, determining the probability distribution over actions

\mathbb{E} : The expectation

This function measures the degree of merit to be in a state s under a given policy, in terms of the expected cumulative reward.

The Action-Value function:

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$

With

$Q_{\pi}(s, a)$: The expectation of the return that can be achieved by taking an action a in certain state s when following a policy π

While the State-Value function evaluates states, the Action-Value function evaluates the quality of actions taken within those states. Both functions are central to value-based RL algorithms.

In addition to learning the value functions, another approach to training the agent is to directly refine the parameterized policy π_{θ} . The fundamental concept underlying this learning process is to increase the probability of those actions that lead to higher returns until an approximately optimal policy is reached (Lehmann, 2024). The probability of taking a specific action under the policy parameterized by θ is denoted as P :

$$P_{\theta}(a|s) = \pi_{\theta}(a|s)$$

A series of observations, actions and rewards within one episode is referred as a trajectory. This trajectory can be expressed as:

$$\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_t, a_t, r_t\}$$

The total reward that the agent accumulates over this episode is the sum of the rewards received at each step of this trajectory:

$$r(\tau) = \sum_0^t r_t$$

The objective is to maximize the expected value of the $r(\tau)$. The objective function is defined as:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [r(\tau)]$$

To maximize $J(\theta)$, the typical approach is to perform gradient ascent. Hence, the gradient over the learnable parameter θ is calculated as $\nabla_\theta J(\theta)$. Each time the policy is updated, the general form of the policy update can be represented as:

$$\theta_{new} \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

where α is the learning rate parameter of the algorithm, determining the step size in the direction of the gradient. By calculating θ , the policy-based algorithms are capable of updating and optimizing its policy directly.

It is also noteworthy that during the learning process, the agent frequently encounters the dilemma of whether to explore unknown states and actions to gain more information about the environment and potential rewards, or to exploit the information already collected to optimize its strategy. This dilemma is referred to as the exploration and exploitation trade-off.

3.3.2 Classification of RL Algorithms

Research on RL has a long history. In 1956, Bellman introduced the dynamic programming approach, which is one of the cornerstones of RL (Bellman, 1956). The classic Q-learning algorithm was proposed in 1989 (Christopher Watkins, 1989). In 2014, the first deterministic policy gradient algorithm was introduced, significantly influencing the development of RL (Silver et al., 2014). With the evolution of Deep Learning (DL), the integration of DL with RL became feasible, and DRL has gained popularity as an alternative to heuristic algorithms. For example, visual input-based RL was applied to vehicle control through a framework known as deep fitted Q-learning (Lange et al., 2012).

Compared to traditional algorithms, DRL offers significant advantages, including the ability to train models with generalization capability through trial and error for complex problems. Well-trained DRL models can be applied directly to subsequent tasks without additional training, thereby saving substantial time.

Over many years of development and research, RL has undergone significant advancements, resulting in the emergence of diverse algorithmic approaches. A comprehensive understanding of the classification of RL algorithms is crucial for gaining insights into their respective strengths, limitations, and suitable application domains. Based on different criteria, RL algorithms can be categorized into several classes. Three of the most commonly used classifications are presented here.

1. Classification based on whether the agent learns the value or the policy:

- **Value-Based Methods:** In value-based methods, the agent explicitly learns a value function, which estimates the expected return for each state. This approach relies on the State-Value and Action-Value functions introduced in the previous subsection. Value-based methods are more efficient for storing and computing in discrete state and action spaces. A classic example is Q-learning, in which the agent stores all state-action pairs in a Q-table, learns the action-value function $Q(s, a)$ by referencing this table, and derives its policy by selecting actions that maximize Q.
- **Policy-Based Methods:** In policy-based methods, the agent directly learns a policy that maps states to actions without necessarily learning a value function. Policy-based methods are well-suited to handling more complex and continuous action spaces. A well-known policy-based algorithm is REINFORCE (Williams, 1992), where the agent directly learns the optimal policy using gradient descent, rather than explicitly estimates the value functions.
- **Actor-Critic Methods:** Actor-Critic methods represent a synthesis of value-based and policy-based approaches. The actor takes action based on the policy, while the critic evaluates the current policy by estimating the value function. The actor then uses this feedback to refine the policy directly. PPO (Schulman, Wolski, et al., 2017) is a prominent example of an actor-critic algorithm.

2. Classification based on the consistency between data collection policy and learning policy:

- **On-Policy Methods:** On-policy methods utilize the same policy for both interacting with the environment and learning from the collected data. Which implies that the policy being improved is also the one used to generate the experience. On-policy methods often yield superior stability and consistency, as the policy is improved more directly and effectively, increasing the likelihood of converge. On-policy algorithms like State-Action-Reward-State-Action (SARSA) (Rummery & Niranjan, 1994) and PPO are widely used.
- **Off-Policy Methods:** Off-Policy methods employ two different policies: a behavior policy for interacting with the environment and collecting data, and a target policy for learning and improving. The target policy is typically the optimal policy that the agent is attempting to learn. Off-policy algorithms offer better sample efficiency and broader exploration capability, thereby assisting in the avoidance of local optima. Examples of off-policy algorithms include Trust Region Policy Optimization (TRPO) (Schulman, Levine, et al., 2017) and Q-learning, along with various extensions based on it.

3. Classification based on whether the agent learns the model of the environment:

- **Model-Based Methods:** Model-based methods entail learning a model of the environment, often represented as a state transition function, which the agent employs to plan and make decisions. These methods predict the outcomes of actions and use this predictive capability to optimize the decision-making. An example of a model-based approach is Model Predictive Control (MPC) (García et al., 1989).
- **Model-Free Methods:** Model-free methods, in the other hand, do not require the learning of an explicit model of the environment. Instead, they learn directly from the observed consequences of interactions with the environment. This approach focuses on learning the value functions or policy based on trial-and-error experiences. Most approaches, including Q-learning and PPO fall under the category of model-free RL.

Additionally, RL can also be classified according to various other criteria, such as online versus offline methods, continuous versus discrete action space, single agent versus multi-agent, and whether the model for action selection is a neural network or

not. In consideration of the distinctive characteristics of the algorithms and the particular requirements of the conflict resolution issue we are investigating, our primary focus is on the algorithm PPO. PPO is an actor-critic, on-policy, model-free, online DRL algorithm. It is applicable to both discrete and continuous action spaces and is suitable for both single agent and multi-agent environments.

3.3.3 Proximal Policy Optimization

Proximal policy optimization (Schulman, Wolski, et al., 2017) was first proposed by OpenAI and has become one of the most widely applied algorithms in the field of RL. PPO is a policy gradient-based algorithm designed to offer a more stable and efficient approach to policy optimization. It is built on earlier policy-based methods, particularly TROP, and overcomes some of their limitations, such as computational complexity and implementation difficulty. As a result, PPO demonstrates robust performance and is relatively simple to implement across a variety of tasks.

The various policy-based algorithms are based on the same principle as outlined in the subsection 3.3.1, but employ different mathematical techniques to perform and optimize the updating of the gradient. The central innovation of the PPO algorithm is its approach to making the learning process more stable by constraining the extent of policy updates. This is achieved through a method known as the clipped surrogate objective, which is formulated as follows:

$$L^{CLIP}(\theta) = \widehat{\mathbb{E}}_t[\min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t)] \text{ (Schulman, Wolski, et al., 2017)}$$

With:

$L^{CLIP}(\theta)$: is an alternative objective function to approximate the real objective function need to be optimized.

$\widehat{\mathbb{E}}_t$: is the estimated expectation value

$r_t(\theta)$: is the probability ratio

\hat{A}_t : is an estimator of the advantage function at timestep t .

ϵ : is a hyperparameter, usually a small positive value that defines the range of clipping.

Directly optimizing the objective function $J(\theta)$ usually leads to destructively large policy updates. To address this problem, the “surrogate objective” $L^{CLIP}(\theta)$ is introduced by constructing constraints.

The probability ratio $r_t(\theta)$ is calculated as:

$$r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)}$$

θ_{old} is the vector of policy parameters before the update, which means $r(\theta_{old}) = 1$.

The advantage function is a metric that measures the “advantage” of taking a particular action a_t in a particular state s_t compared to the average action at that state under the current policy θ . It provides a means of quantifying whether an action taken at time t is more or less advantageous than what the policy would typically suggest. It is formally defined as:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

The advantage estimate \hat{A}_t is an approximation of the advantage function A . It should be noted that when A is positive, this indicates that the action a_t taken in state s_t is preferable than the average action in that state. Consequently, the policy should be updated with the aim of increasing the probability of selecting the action a_t in the future. Conversely, when A is negative, the action a_t is deemed to be inferior to the average, thus the policy should be adjusted to decrease the probability of selecting that action.

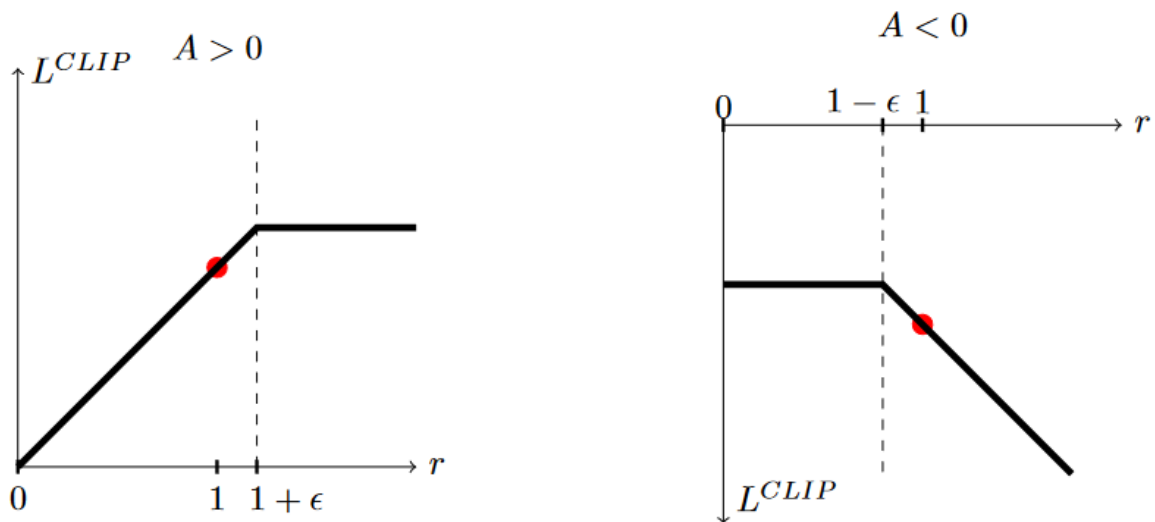
The clip function with three terms here means:

$$\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) = \begin{cases} 1 - \epsilon & \text{if } r_t(\theta) < 1 - \epsilon, \\ r_t(\theta) & \text{if } 1 - \epsilon < r_t(\theta) < 1 + \epsilon, \\ 1 + \epsilon & \text{if } r_t(\theta) > 1 + \epsilon. \end{cases}$$

Essentially, it limits the probability ratio $r_t(\theta)$ to the range of $[1 - \epsilon, 1 + \epsilon]$. After the clip of the r , the minimum of clipped and unclipped objective $r \hat{A}_t$ is taken, so the final objective is always a lower bound.

Figure 3.4 shows a single timestep of the surrogate function L^{CLIP} as a function of the probability ratio r , for positive advantages (left) and negative advantages (right). If $A > 0$, the surrogate objective L increases as the action a becomes more likely. The policy parameters θ are adjusted to reflect this change. By clipping the ratio r , this positive effect on the objective function disappears once we move outside the clip range. This clipping process is conservative, only applying the clip if the objective function would be improved. If the policy changes in the opposite direction that decreases L , the ratio r is not clipped due to the minimum function in the equation, resulting in the actual value range of L^{CLIP} in this case being: $(0, (1 + \epsilon)A)$. Conversely, if $A < 0$, the value

range of L^{CLIP} is $(-\infty, (1 - \epsilon)A)$. To put it another way, the clip in probability ratio $r_t(\theta)$ is ignored when it would worsen the objective and included only when it would improve it. This approach prevents extreme "optimization" due to a single data point, mitigating the risk of drastic degradation in policy performance. The red circle on each plot shows the starting point for the optimization. By employing the clipping mechanism, PPO simplifies the complex constrained optimization process, reducing computational overhead and making it easier to implement and tune.



• Figure 3.4: The clipping of PPO's surrogate objective function (Schulman, Wolski, et al., 2017)

Apart from clipped surrogate objective, PPO can also employ a penalty on Kullback-Leibler (KL) divergence with an adaptive penalty coefficient as an alternative. However, the clipped surrogate objective is generally preferred for its simplicity and effectiveness, and therefore most commonly associated with PPO. The details of the adaptive KL penalty coefficient and the more complex mathematical calculations implicit in the PPO algorithm are not covered here.

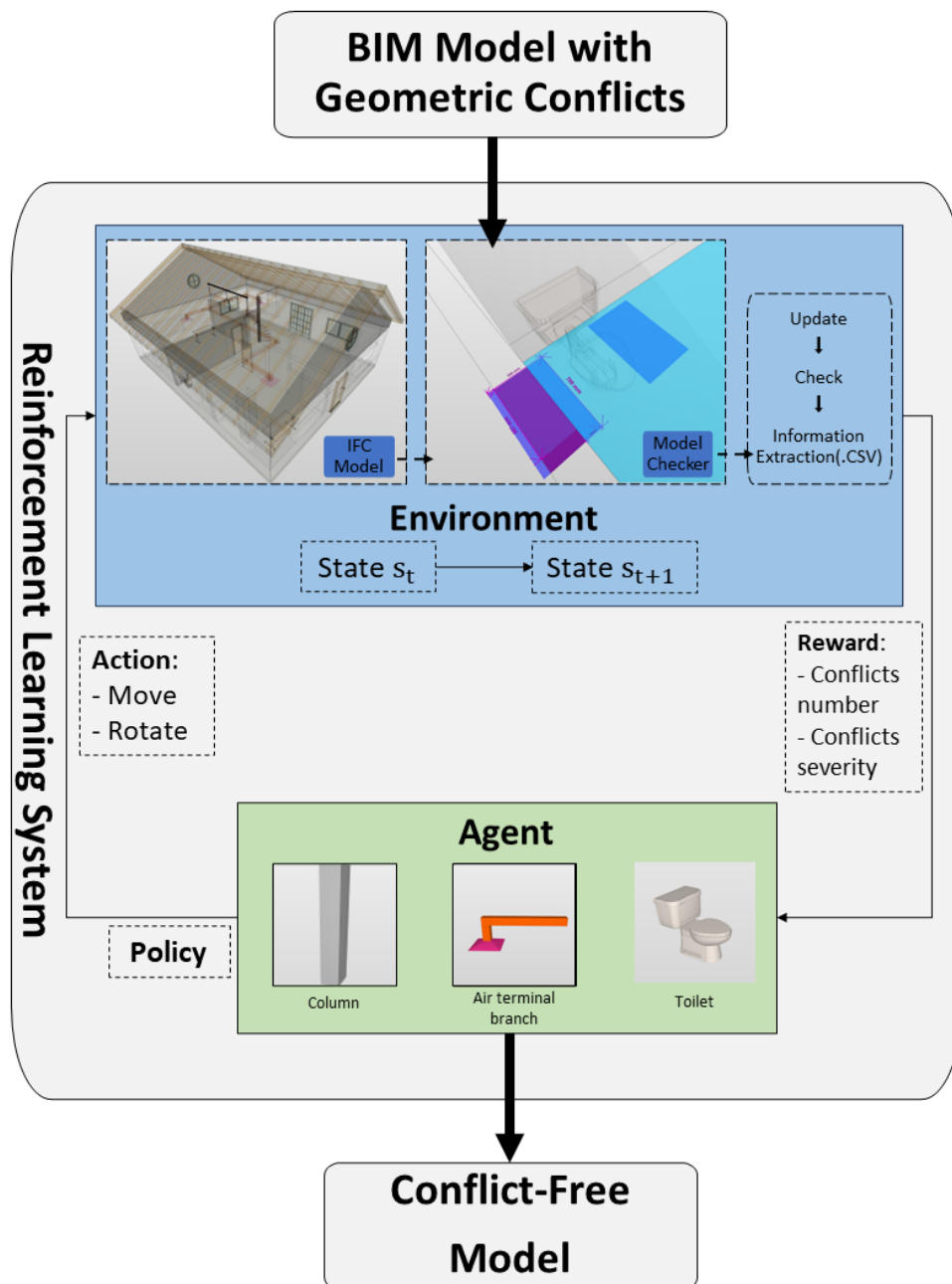
In addition to the core concept of clipping the optimization, PPO exhibits several other noteworthy characteristics compared to other RL algorithms:

- PPO allows policy updates to be performed multiple times in each iteration using the same data set, which improves data efficiency.
- PPO can be easily parallelized, allowing it to be applied to large-scale problems.
- PPO is capable of effectively balancing exploration and exploitation, making it suitable choice for complex environments where other methods may encounter difficulties in converging or require extensive tuning.

In conclusion, PPO represents a substantial advancement in the field of RL. By maintaining the updated policy “proximal” to the previous one, PPO ensures stability in the learning process and address the limitations of previous policy gradient methods. This has made it a popular choice for training agents in diverse environments, from simulated games to real-world robotic tasks, since its introduction. Consequently, it is the RL algorithm that we employ in this research.

4 Methodology

This chapter presents our approach, which applies RL to automatically resolve geometric conflicts in a BIM model. Our methodology integrates the BIM model and a model checker into the RL environment, thereby facilitating the training with real BIM. Figure 4.1 illustrates the proposed framework.



• Figure 4.1: The framework of this study

Fig.4.1 displays the pipeline of our approach. The process commences with an existing BIM model that contains various geometric conflicts. The IFC file is firstly imported into the model checker, which is integrated into the established RL environment where the agent operates. The RL training loop proceeds as follows: the agent takes action to reposition (move or rotate) the conflicted components within the BIM model. Subsequently, the environment is updated in response and based on the checking results of the integrated model checker, the agent receives reward and updates its policy accordingly. The iteration continues until the environment reaches a terminated state, indicating that the BIM model has passed the conflict check. At this point, the conflicts have been successfully resolved, and the final conflict-free BIM model is exported as the resolution. The RL agent learns the optimal conflict resolution strategy through this training progress, with the objective of minimizing both the number and the severity of conflicts reported by the model checker. The trained RL model is saved for further evaluation and testing. This chapter will detail how the conflict resolution task is modeled in the form of RL, including the construction of state, action, and reward modules based on the OpenAI Gym framework (Brockman et al., 2016) and the application of the PPO algorithm as the underlying engine for the agent.

4.1 State Module

In the context of RL, the concept of state is fundamental to the decision-making process of an agent. The state represents the current situation of the environment with which the agent interacts. The accuracy and completeness of the state representation are crucial, as they directly influence the agent's capacity to predict future states and outcomes, and consequently, to learn an optimal policy. In this study, the environment is constructed using the Gym framework, which is a Python library for the creation and manipulation of RL environments. The modularity and flexibility of Gym facilitate the creation of custom environments tailored to specific research needs. Its compatibility with various RL libraries serves to further enhance its utility, making it a powerful tool.

In Gym, the term “observation” is often used interchangeably with “state”. However, they are not always identical. An observation is the data that the agent receives from the environment at each step, which may be a full or partial representation of the underlying state. In many practical scenarios, the observation only provide partial key information, meaning that the agent must learn to infer or approximate the true state of the environment from the observations due to the high level of complexity involved.

In contrast to other studies that construct an abstracted and highly simplified building space and building model comprising only a few relevant elements for training purposes, our research integrates a complete IFC model into the RL environment. In a real BIM model, the relocation of a single element may have ramifications for numerous adjacent components. Consequently, a multitude of regulations and constraints must be considered. By employing a real IFC model, our approach addresses the intricacies and interdependencies intrinsic to actual building designs. This signifies that the state encompasses all components and semantic information within the BIM model. However, it is impractical to use state to represent the complete environment. Therefore, based on the findings presented in section 2.2, we focused on: (1) information about the conflicts and (2) the components causing these conflicts. This knowledge about the conflicts and the properties of the components is extracted as observations. The parameters of the observation space design are summarized in the following table:

Key parameters		Parameter definition
number of conflicts		The total number of conflicts that we aim to resolve in the BIM model.
severity of conflicts		The accumulative severity indicator for conflicts that we aim to resolve in the BIM model.
number of created conflicts		The total number of conflicts, that we do not intend to cause, but arise during the iteration due to inappropriate action.
severity of created conflicts		The accumulative severity indicator for created conflicts that we do not intend to cause
element type	element 1	The IFC class of the element.
	element 2	
element rotation	element 1	The indicator for the current direction of the element.
	element 2	
element vertices	element 1	The calculated world-coordinates of the eight vertices of the bounding box of the element
	element 2	

Table 4-1: Summary of the observation parameters

Number of conflicts represents the total number of existing conflicts in the BIM model. The ultimate goal of the RL agent is to reduce this number to zero.

Severity of conflicts provides a measure of the overall severity of conflicts within the model. Each conflict is categorized into three levels of severity: Low, Moderate, and Critical. In order to quantify the severity in the RL environment, the three levels are represented with the numbers 1, 2, and 3, respectively. The size of the number indicates the severity of the issue, and the sum of these values helps in assessing the cumulative impact of conflicts, thus providing a complement to the number of conflicts.

Created conflicts refers to new conflicts that are introduced by the agent's actions. These conflicts are tracked separately because they indicate whether the agent's actions inadvertently worsen the situation. The agent is trained to not only resolve existing conflicts but also to avoid creating new ones. An effective policy should lead to zero created conflicts alongside resolving initial conflicts.

Element type identifies the type of elements involved in the conflicts. Different IFC classes are assigned specific numbers for easy identification and processing in RL. Based on the conducted literature review, the type of the involved element can greatly influence the severity and common resolution of this conflict, and guides decisions on which element regarded as the agent to move or adjust.

Element rotation describes the orientation of the element as a matrix, with rotations limited to multiples of 90 degrees. Simplifying rotation to discrete angles helps in managing the complexity of geometric representation.

Element Vertices represents the vertices of the element's axis-aligned bounding box in the world coordinate system. It provides geometrical information about the element's size and placement. Given that different objects have different shapes, and that the number of real vertices can vary significantly, especially for round objects, the vertices of the element's bounding box are calculated and included in the observation space. Along with rotation, they represent the most important geometrical information about the element.

A conflict in BIM model can be related to one or more elements. In the case of a single element, it is typically the attributes and properties that fail to meet design regulations, which is not within the scope of this research. For other issues, two elements are typically engaged in a state of geometric conflict. Even when more than two elements are in a single conflict at the same time, they can be broken down into two-by-two conflicts.

Accordingly, the element type, rotation and vertices parameters are considered with regard to two conflicting elements, respectively.

With the exception of the element vertices, all listed parameters are formulated as discrete numbers. These are collected and flattened to a one-dimensional vector as the observation at each time step for the RL algorithm. The specific implementation details for collecting these parameters using model checker APIs and IfcOpenShell are outlined in the following section.

4.2 Action Module

To successfully resolve conflicts within BIM models, actions should be formulated in a way that allows the agent to directly manipulate the elements in the model in ways that are practical and effective, thereby addressing the identified conflicts. Based on the analysis of the literature and practical experience, the primary actions involved in resolving geometric conflicts generally involve moving or rotating one of the conflicted elements, and in some cases, both.

Available action		Action index	Illustration
Move	Positive x: right	0	
	Negative x: left	1	
	Positive y: forward	2	
	Negative y: backward	3	
	Positive z: up	4	
	Negative z: down	5	
Rotate	Clockwise 90°	6	
	Anticlockwise 90°	7	

Table 4-2: Summary of the action

An increase or decrease of 10 mm in the x, y, and z coordinates within a three-dimensional coordinate space signifies one movement of the elements in disparate directions, including right, left, forward, backward, up, and down. The value of 10 mm is determined because in most BIM software, it is the default tolerance of the clash detection. Given that the element's orientation is constrained to four directions in the observation space, the available actions are limited to rotations of 90 degrees in either a clockwise or anticlockwise direction, centered on the z-axis.

Eight available actions are designed in the RL environment to construct a discrete action space. Each possible action can be indexed and encoded as part of the action space. These actions are summarized and illustrated in Table 4-2. They are performed by a Python library, *IfcOpenshell*, which will be introduced in the subsection 5.2.

4.3 Reward Module

The reward system functions as the primary feedback mechanism, indicating to the agent whether its actions are leading to improvements or deteriorations in the state of the environment. This feedback is of paramount importance for the agent's learning process, as it utilizes the rewards to evaluate the effectiveness of its actions and to refine its policy to maximize long-term rewards. In the proposed RL system, the reward is designed with three main considerations based on the state of the current BIM model, mainly related to the output of the model checker:

1. The change in the number of conflicts
2. The number of created conflicts
3. The change in conflicts' severity

The number of conflicts is at its highest at the beginning of the training. As the iteration goes on, the number of conflicts decreases gradually until it reaches zero, with each conflict number decreased:

$$reward = 1$$

Should the agent's actions result in the reintroduction of previously resolved conflicts, a penalty of $reward = -1$ is applied. This discourages the agent from taking actions that undo previous progress.

During the training process of changing the geometric position of the elements, it is likely that new conflicts will arise, such as the agent intruding with another object around or the agent moving out of the original space. In such instances, each new conflict results in a penalty of

$$reward = -1$$

Given that we are modeling in a digital BIM environment instead of the physical world. The conflict with other elements during the process is acceptable as long as it is resolved at the end the training. Therefore, in the case of a newly created conflict is subsequently resolved, an equal reward $reward = 1$ will be granted. This reward encourages the agent to correct any additional issues that it may have introduced.

As outlined in section 4.1, conflicts are classified according to their severity, which is measured on a three-point scale: low, moderate, and critical. The severity is quantified as a number, which is used to compare when the conflict number remains constant. This indicates either a minor improvement or regression in the model. For each unit decrease in the severity score:

$$reward = 0.2$$

This enables the agent to focus not only on the quantity of conflicts but also on their gravity. Similarly, should the number increase by one, whether in regard to existing conflicts or created conflicts, a punishment of $reward = -0.2$ will be administered.

In essence, the designed reward system guides the agent in learning to make decisions that consistently lead to better states, which is evidenced by a reduction in the number or severity of conflicts, thereby achieving the overarching goal of the task. In order to conduct comprehensive checking of the model and to reflect the state of the model within the reward system, the rulesets in the model checker are predefined align with the aforementioned three considerations. The configuration of the rulesets takes three aspects into account:

1. The primary rule to check specific types of conflicts that the agent aims to resolve.
2. The general BIM validation rulesets ensure that the complete model is comprehensively taken into account, including the implicit structural rationality
3. The additional rules to detect the specific conflicts that may arise during the relocation of an element according to its type and properties.

The subsequent chapter will provide a comprehensive account of the implementation of rulesets in the model checker, along with a detailed exposition of the extraction and processing of the checking results, which are integral to the proper functioning of the reward system.

4.4 PPO-based Conflict Resolution Algorithm

To summarize the methodology, the resolution algorithm, which is the main contribution of this paper is presented. The PPO is selected as the policy optimization algorithm, guiding the RL agent to take actions from the action space. The structured summary of the proposed PPO-based resolution algorithm is described as follows:

Algorithm PPO-based conflict resolution

Reset Environment: generate new environment with a new IFC model

for each conflict \in conflicts list **do**

while steps < maximum_steps_per_episode **do**

 steps = steps + 1

 action: select action from action space using PPO policy

 update: perform the action and modify the IFC model

 check: check the state of the IFC model with the predefined rules

 result: extract key parameters from the result of model checker

 reward: calculate reward based on the reward system

if the number of conflicts decreased **then**

 Break while

end if

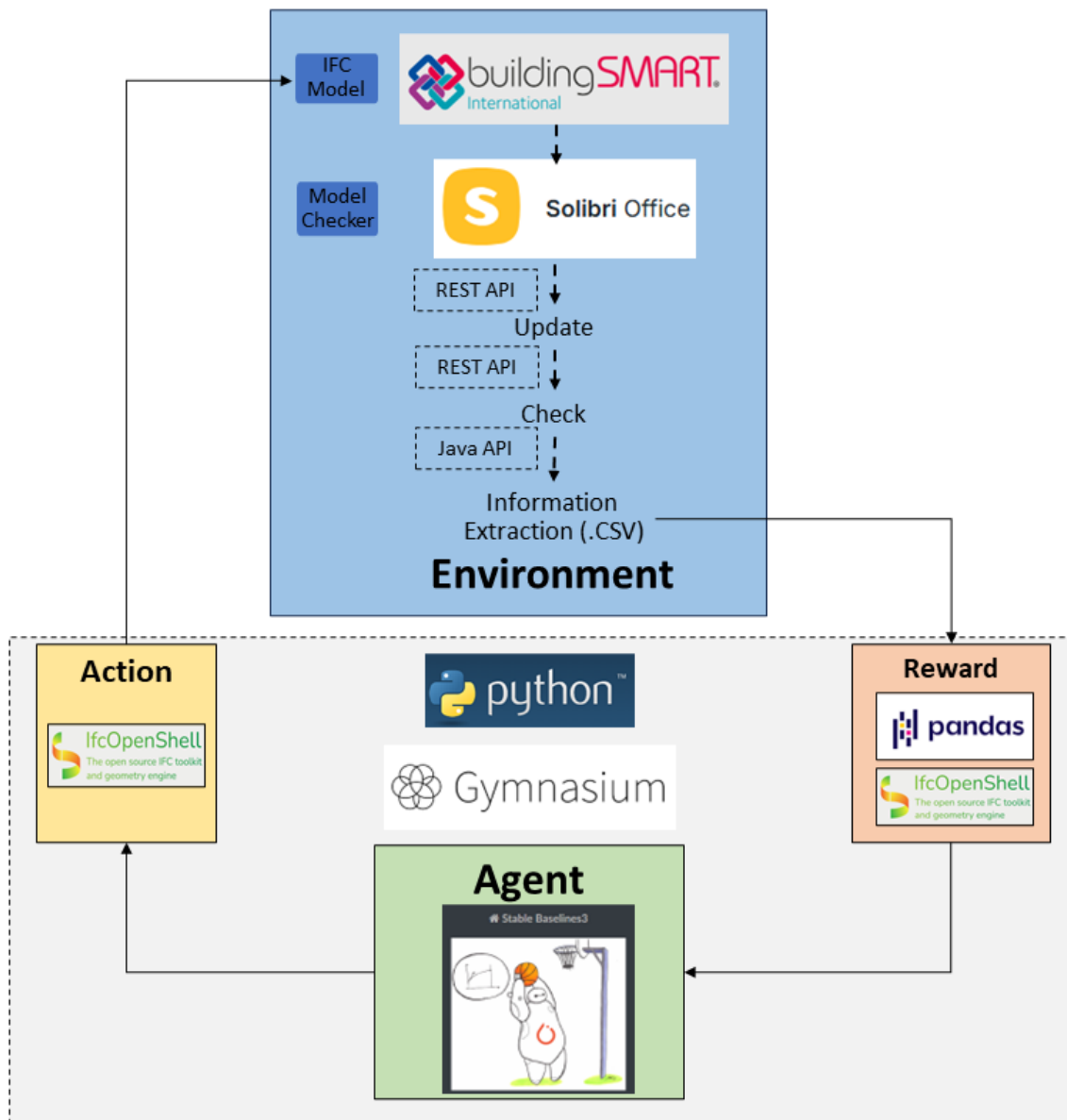
Else: Truncated

end for

The primary loop of the code iterates over a predefined number of episodes, with each beginning by resetting the environment. Upon each reset, the BIM model is reinitialized with a new set of conflicts, thereby ensuring that each episode starts from a consistent state and maintains the variability of the training environment. At each step, the agent attempts to resolve one conflict by applying an action selected by the PPO policy, generating a new state of the IFC model, which is then evaluated by the integrated model checker. The observation received is then utilized to calculate the reward, which guides the agent's learning process and policy updates. Upon the resolution of a conflict, the algorithm proceeds to address the next conflict in the sequence. This iterative process continues until all conflicts have been resolved, signaling the termination of this episode with the environment marked as "done". Alternatively, the episode may truncate when the maximum step limit is reached. Details regarding the hyperparameters and training logging settings will be provided in the next chapter. Further experiments and three use cases will be presented in chapter 6.

5 Implementation Details

This chapter provides a detailed description of each technical implementation of the proposed methodology, which employs a combination of BIM-based model checker APIs and various Python libraries. The full implementation details of the RL system are illustrated in Figure 5.1:



- Figure 5.1: Implementation of the RL system

5.1 Model Checker

Solibri Office was selected as the model checker for this research. It facilitates the import of building models from various BIM authoring software through the standardized IFC interface and offers an advanced and comprehensive library of customizable rule templates to address the most demanding quality assurance tasks (Solibri, 2024).

5.1.1 Solibri Rulesets

Solibri provides a large quantity of pre-implemented rulesets, which can detect not only geometric clashes, but also semantic errors in the model across various disciplines, including architecture, structure and MEP. It is noteworthy that the rulesets are highly extensible. The parameters can be adjusted to align with particular requirements, such as filtering component types or establishing distance tolerances. Moreover, the rulesets can be augmented through the utilization of Solibri Java API, which enables the development of custom rulesets tailored to specific needs. Compared to design check functionalities in a self-formulated environment, this degree of comprehensiveness and adaptability ensures that any unintended conflicts caused by relocating elements during training can be identified, thus maintaining the integrity of the BIM model.

The completed and accurate rule selections and setup can maximize the benefits of the application of the model checker and ensure the optimal functioning of the RL reward system. Conversely, the rules should not be excessively repetitive and should be tailored to the specific objective, in alignment with the aforementioned three principles in section 4.3. Figure 5.1 provides an example of selected rulesets in Solibri for the use case of a column positioned in front of a window. In this particular instance, three distinct types of rulesets are selected accordingly. The "Clearness in Front of Windows" ruleset is the primary means of detecting the focused conflict that the agent aims to resolve. Subsequently, a series of general rules are employed to ensure comprehensive compliance with the integrity of the entire building, including an assessment of its implicit BIM rationality. The "Components Above Columns" rule ensures that when the column is relocated outside of the building, the model checker can indicate that a conflict indeed exists, even when the clearness of the window is guaranteed and there is no collision between the column and other components.

Name	Support Tag
▼ Rulesets Open in Solibri	
▼ BIM Validation - Architectural	
▶ Model Structure Check	
▶ Component Check	
▼ Clearance	
§ Clearance in Front of Windows	SOL/226/3.1
§ Clearance in Front of Doors	SOL/226/3.1
§ Clearance Above Suspended Ceilings	SOL/222/4.2
§ Free Area in Front of Fixed Furnishing	SOL/226/3.1
▼ Deficiency Detection	
§ Required Components	SOL/11/4.2
§ Unallocated Areas	SOL/202/1.4
▼ Components Below and Above	
§ Components Above Columns	SOL/23/5.2
§ Components Below Columns	SOL/23/5.2
§ Components Above Beams	SOL/23/5.2
§ Components Below Beams	SOL/23/5.2
§ Components Above Walls	SOL/23/5.2
§ Components Below Walls	SOL/23/5.2
§ Revolving Doors Must Have Swinging Door Next to It	SOL/222/4.2
§ Slabs must be Guarded against Falling	SOL/236/1.2
▶ General Space Check	
▼ Intersections Between Architectural Components	
▶ Intersections - Same Kind of Components	
▼ Intersections - Different Kind of Components	
§ Door Intersections	SOL/1/5.0
§ Window Intersections	SOL/1/5.0
§ Column Intersections	SOL/1/5.0
§ Beam Intersections	SOL/1/5.0
§ Stair Intersections	SOL/1/5.0
§ Railing Intersections	SOL/1/5.0
§ Suspended Ceiling Intersections	SOL/1/5.0
§ Wall Intersections	SOL/1/5.0
§ Slab Intersections	SOL/1/5.0
§ Roof Intersections	SOL/1/5.0
▶ Intersections of Furniture and Other Objects	

- [Figure 5.2: An example of the predefined rulesets in Solibri](#)

Once the rules have been selected, the implementation of a Python-based update and checking progress, as well as the incorporation of the checking results into the RL training environment represent a crucial aspect of our approach. The realization of this process is made possible by leveraging Solibri APIs.

Solibri offers the possibilities of different kinds of API, and in this study, the integration relies on both REST API and Java API. The objective is to achieve the following key functionalities:

- Update the modified IFC model continuously
- Check conflicts within the IFC model
- Extract results for further interpretation in Python

5.1.2 Solibri REST API

The Solibri representational state transfer API facilitates the first two tasks. This type of API allows for easy integration with Python, making it well-suited for our needs. To activate the REST API, the Solibri software must be launched with special command line arguments:

Name	Arguments	Description
--rest-api-server-port	port number, for example = 10876	Open Solibri with REST API in the given port number
--rest-api-server-local-content		Show local content, like full paths for opened model, instead just model name
--rest-api-server-http		Launch Solibri with http connection instead of https

Table 5-1: The commands for launching Solibri using REST API

To meet our objectives, specific REST requests provided by Solibri are selected:

- `/models/{modelUUID}/update` (PUT) - This request is used to update the existing IFC model on the Solibri server. By sending a PUT request with the new IFC file, the server updates the model accordingly. The PUT method is idempotent, meaning that making the same request multiple times will yield the same result without creating duplicate resources. Universally Unique Identifier (UUID) is a 128-bit label used for information in computer systems, and the `{modelUUID}` path parameter is essential, as it uniquely identifies the specific resource on the Solibri server. It must be obtained through another request before the training loop begins:
 - `/models` (GET) - This request retrieves the current models, allowing us to obtain the necessary UUID. The model UUID remains consistent throughout the training process.
- `/checking` (POST) - This request initiates a model check, utilizing Solibri's comprehensive rulesets and returns the results.

However, the response of REST API checking provide limited information, particularly lacking the elements' Globally Unique Identifier (GUID), which is essential for identify elements within the IFC file. Due to the current limitation of the Solibri REST API functions, we have employed the Solibri Java API to achieve seamless integration with the Python and the Gym environment. This offers more detailed access to the model data and allows us to extract the required information, including the GUIDs.

5.1.3 Solibri Java API

To facilitate the extraction of necessary information from Solibri following each model check, a Java file named “CHECKING EXPORTER” has been programmed based on the official templates provided by Solibri. This file will be executed each time the model is checked in Solibri, with the objective of retrieving essential data for the RL environment.

Following the installation of the Java file into the Solibri software via Maven, a preliminary configuration is necessary prior to the inaugural utilization of the software. To commence the Solibri software, navigate to the VIEWS tab and select the CHECKING EXPORTER. A new window with the same name will appear, indicating that the tool is ready for use. Once configured, the CHECKING EXPORTER runs automatically with each model check conducted in Solibri. The Java API extracts specific information regarding issues detected during the check and exports this data into a Comma-Separated Values (CSV) file for subsequent analysis. This process guarantees the availability of all relevant conflict data for integration into the RL environment. Each clash instance comprises the following details:

1. IFC GUID of conflicting components
2. name of the rule violated
3. severity of the conflict

A snippet of the exported CSV file, obtained after performing the check with the selected rules, as illustrated in Figure 5.2, is displayed in Figure 5.3:

Component	Rule	Severity
OC8srG6W1AKwwePetp1Wac;2J161QV4fABgMWjCtsmots	Clearance in Front of Windows	CRITICAL
OwhDnDIi9DpAR7UIyTmBL1;0AbzXzgCL60PX\$tGp8dg6z	Clearance in Front of Windows	CRITICAL
1p6FQoXwz4CRGg\$SYF2PEb;1ZBjN7PoT2MvzdRcl1V1BTU	Clearance in Front of Windows	CRITICAL
0k61t0mRHBmA62EJuvvUuc;2T65n4KE1CDe_qEEHoeJGv	Clearance in Front of Windows	CRITICAL
0w\$7P_3av7Ah4ZEtCM11IL;3nQVgONdnAD8_3nXa3owdi	Clearance in Front of Windows	CRITICAL
074DmBRGz6Gw1UzFTwmw1k;3bUjplZbL8Px81PHzVr\$I0	Clearance in Front of Windows	CRITICAL
00ycwsm5r3jBfRVHoHMMCr	Spaces Must Have Doors	MODERATE
0q6C7UpXLDzQ2Tqs\$EvRZk	Spaces Must Have Doors	MODERATE
1ECVuFS0jC\$uXcVWNyD8QS	Spaces Must Have Doors	MODERATE
38ue4gr1r4PRDzZ2zpgGr6;0InCfUMeH4cACy\$f1Tz4s_	Wall - Wall Intersections	CRITICAL
3tggTTVwbBsuBLDdRHmmNS;060aN4G3vFhvE42FpJLlqT	Wall - Wall Intersections	CRITICAL
23hVperYvEK81H0SZ\$2\$e0;2011KAC8z5ixG0mBsgs_Yv	Beam - Beam Intersections	CRITICAL
OgCHwFSLP8IAGEJ84\$PkNy;0A_1_4CnXCIvAQxE7w8qgt	Beam - Beam Intersections	CRITICAL
1ec8hzrMD290_AaNi\$C\$BR;1MwOC189f2Qupxgz_WPou4	Beam - Beam Intersections	CRITICAL
3fLw\$cNwH1g0zVcxo0h61m;1s67JsZKD9RQ3WpL\$vkZz	Beam - Beam Intersections	CRITICAL
2EsbTMOKLCMho9IZTToA7JI;15ua6s3hr90PcWjp3Rbkdd	Window - Window Intersections	CRITICAL
3Hbo6CEcv3Kx9kETNBdPRm;2uv1FZmBrAg92PXwSM6vmn	Window Intersections	LOW
38ue4gr1r4PRDzZ2zpgGr6;3TV8LZdcP100zgb10fSHTi	Window Intersections	CRITICAL
1fdYw1aZDB2v5ZHPPT6kQM;1GrjhZbpP7dvTRBtSE3agH	Window Intersections	CRITICAL
2tFgifiNwBw9yv7A\$RSzgy;3VcWYwMu13PPj1_3YGBkEa	Window Intersections	MODERATE

• Figure 5.3: An example of the extracted checking results

Solibri automatically assigns a severity to every issue based on predefined criteria, such as the specific category of the result (e.g., incorrect door opening direction), or the degree of deviation from a given requirement (e.g., too close a column to a window). Additionally, it can be customized through the use of severity component filter parameter tables within the rule parameters, allowing users to tailor the evaluation of the severity according to their specific needs.

These three types of information are specifically selected for two primary purposes:

1. **Extracting IFC model information:** The data is exported to a CSV file, which is readily readable and manipulable using Python's pandas package. Each row in the CSV file corresponds to a single conflict in the IFC model, providing a clear indicator of the model's current state during the optimization process. The distinct rule names and severity levels indicate the extent of the rule violations and are directly correlated to the reward module.
2. **Retrieving element profiles:** The GUIDs of the conflicting components are utilized as input into the `IfcOpenShell` toolkit to extract comprehensive profiles of the components. These profiles encompass the component's IFC type, orientation state, and vertices. Based on the profiles, the proposed RL system can observe the changes of the model, which are then applied to update the observation for the agent.

Prior to the commencement of training, the initial settings in Solibri must be completed manually. These include the implementation of the Java API, the classification of building elements, and the selection of appropriate rulesets for checking. Once the initial configuration is complete, the entire training process is automated through the use of Python scripting. This automation handles model checks, model updates, and the exportation of certain check results functionalities in the framework.

5.2 IfcOpenShell

To construct an RL environment for the BIM model that permits the programmatic reading and modification of IFC files, we employ a library named `IfcOpenShell`. `IfcOpenShell` serves as a versatile digital tool, spanning from design to construction, that supports our objectives. This open-source C++ software library is designed for working with the IFC file format, and it also provides good Python bindings to the core C++ system, as well as high level analysis and authoring functions (`IfcOpenShell-Python`, 2024). The full capabilities of the C++ core are available in Python, making `IfcOpenShell` an ideal

choice for seamless integration into our environment. `IfcOpenShell` supports a wide range of operations within IFC files. In this study, we employ it to achieve two core goals:

- **Accessing geometric and semantic Information:** `IfcOpenShell` enables the extraction of specific geometric and semantic details about building components, utilizing the GUID obtained from Solibri APIs. This information is crucial for the RL observation module.
- **Manipulating components:** The library provides the functionality to move and rotate building components within the IFC model. This capability is essential for implementing the actions chosen by the RL agent, allowing the system to modify the environment in response to the agent's decisions.

Through these functionalities, `IfcOpenShell` plays a critical role in bridging the gap between the BIM model and the RL environment, enabling dynamic interaction and real-time updates within the system.

5.3 Stable Baselines3

Stable Baselines3 (SB3) is a set of reliable implementations of RL algorithms in PyTorch. In comparison to developing PPO from scratch, the algorithms offered by SB3 are highly optimized and well-tested, ensuring reliable and consistent performance across various environments. Moreover, the library facilitates seamless integration with custom environments made using the Gym framework. This enables us to prioritize research and practical applications over the low-level algorithm implementation details.

Furthermore, SB3 incorporates a multitude of wrappers and utilities that significantly enhance the flexibility and efficiency of workflows. These facilitate a seamless customization of the environment or action, data processing, and monitoring and logging of training process. The built-in support for TensorBoard is crucial for our research, as it enables the easy storage, visualization, and assessment of training processes. During training, a *CustomCallback* class, derived from the *BaseCallback*, is designed to monitor and log a series of actions, rewards, done states and observations. The *EvalCallback* is responsible for evaluating the model at periodic intervals, which can then be visualized using TensorBoard, and saves the best-performing model based on evaluation metrics. The *CheckpointCallback* ensures that the model is periodically saved, thus facilitating recovery and analysis.

In conclusion, the IFC file and Solibri APIs are employed in conjunction to construct the RL environment. The `IfcOpenShell` library executes the selected resolution action on the IFC model, with the actions guided by the PPO algorithm implemented by SB3. Solibri then evaluates the new state of the IFC model using REST API to perform updates and checks. Observation and reward are extracted using Java API, pandas and `IfcOpenShell` to update the agent's policy based on PPO. These implemented components are tightly integrated, forming a complete and cohesive RL system.

6 The Experiments and Evaluation

This section first introduces the general experimental environment and configurations, after which three proposed real cases are presented. In order to accommodate the specific characteristics of different use cases, the settings in the RL environment and in Solibri are adjusted accordingly. This approach facilitates a more streamlined training process.

6.1 The Experimental Environment and Configurations

The experiments were conducted in two distinct devices. The laptop is equipped with a 12th Gen Intel(R) Core (TM) i5-12500H CPU and 16GB of RAM, while the operating system and Solibri software are Windows 11 and Solibri Office 24.5.0.31, respectively. The second computer is equipped with an Intel(R) Core (TM) i7-7700H CPU and 32 GB RAM, running the Windows 10 operating system and the Solibri Office 24.5.0.31 software. The simultaneous training of two distinct use cases is therefore possible. However, a parallel environment was not constructed, as the vectorization of the RL environment is not viable due to the integration of Solibri.

The employed PPO algorithm is implemented in the SB3 library. The neural network architecture for the policy and value functions is a standard feedforward network, commonly designated as a Multilayer Perceptron (MLP). The network comprises two hidden layers, each comprising 64 units. The Rectified Linear Unit (ReLU) activation function is applied to each hidden layer, providing non-linearity and facilitating the learning of complex patterns in the input data.

The policy network generates a set of action probabilities. In contrast, the value network produces a single scalar representing the expected return from the current state. The parameters of both networks are optimized using the Adam optimizer, with a learning rate of $3 * 10^{-4}$, which facilitates efficient learning while also balancing the trade-off. A batch size of 64 is employed during training, which is a common choice to ensure a balance between computational efficiency and the stability of gradient estimates. The data was split into two batches, and the neural network was updated with 128 samples per batch. The number of epochs was set to four, which helps ensure that the learning process is thorough and that the updates are stable and effective.

It is important to acknowledge that the proposed methodology can be theoretically applied to any conflicts within IFC files that can, in practice, be resolved through the eight available actions that can be performed by IfcOpenShell. However, for the purposes of evaluating the feasibility of this approach, we have only implemented the RL algorithm in three different specific use cases.

The training was conducted in a typical one-family house, as illustrated in Figure 6.1. It comprises 8 entity types, 128 entities, and 659 relations. This standard IFC model is an open-source model offered by IfcOpenShell. To create different conflicts in specific use cases, this IFC file was imported and edited in Revit 2023. Subsequently, it was exported back as an IFC file using the standard IFC 4 MVD: design transfer view.

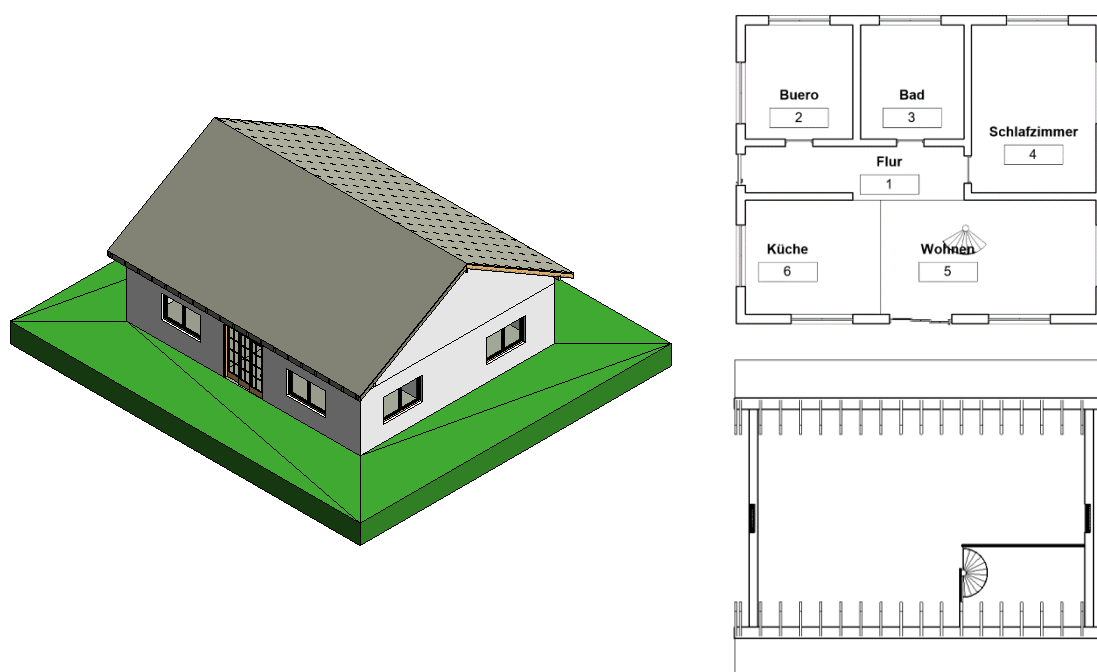


Figure 6.1: The IFC model used for training

6.2 The Toilet-Wall Conflict

In bathrooms, the placement of the toilet seat in close proximity to the wall is regarded as a conflict. The DIN 18040-2 standard specifies a minimum lateral distance of 30cm between the toilet and the wall, as this distance is necessary to ensure accessibility and comfort. Additionally, a shorter distance could lead to alignment issues with the waste pipe of the plumbing system. This issue can be extended to many of the conflicts of insufficient distance between furniture, and inadequate accessibility of spaces, which are very common in the architectural discipline.

6.2.1 The Experiments

The training environment was initially established in Solibri, for the purpose of selecting appropriate rulesets and modifying them to detect the toilet-wall conflicts and other potential issues that may arise during the training process. These modifications were made according to the specific requirements of our custom needs. For instance, if the toilet is not moved in alignment with the wall, but rather at a perpendicular angle, it may potentially resolve the conflict. However, this would result in the toilet being situated in the center of the room, lacking any connection to the surrounding walls. This outcome is clearly untenable. In other scenarios, the toilet may encounter interference from other surrounding fixtures or furniture during its movement, or it may encroach upon the designated bathroom space, even extending beyond the boundaries of the room. Given these considerations, three distinct rulesets were especially selected within Solibri, based on the pre-implemented rulesets:

1. § 26.15 & 26.17 Shower and Bathrooms. This rule was adjusted to check the free space on both sides of the toilet seat. The parameter is set to 500mm on both sides, thereby ensuring that the distance between the toilet seat and the wall is at least 500mm.
2. # 222 Component Distance. A new rule is created under this ruleset in the Solibri ruleset manager to ascertain the proximity of any `lfcWall` elements to the toilet seat. The maximum distance permitted is 10mm, ensuring that the toilet is not positioned in the center of the room.
3. § Object Intersections. The intersection of the toilet seat with other components is checked with this rule.

The complete set of rules is displayed in Appendix A.1. In practice, this type of conflict between a toilet and a wall is typically resolved by moving the toilet seat to an adjacent position along the wall. Accordingly, the action space for this specific use case was designed as a subset of the action space described in subsection 4.2. The available actions are as follows: (1) positive x move, (2) negative x move, (3) positive y move, and (4) negative y move. The distance traversed by each movement is 50 mm. This number was selected because it allows for the resolution of the issue in 10 time steps, assuming the policy is optimal, that is, the toilet seat is moved away from wall beside it. In order to resolve a conflict by moving the element, it is preferable that the movement be as minimal as possible to maintain the original design intent. Consequently,

for each conflict, the iteration lasts for a maximum of 30 steps (*max_steps_per_episode*).

It is also necessary to specify the observation space within the RL environment. In this instance, the element type parameter is set to "IfcFlowTerminal" and "IfcSpace", which indicate the classes of the two conflicting elements in Solibri. Although the conflicted element is not a wall in Solibri, the space is constructed by four walls. And in this case, the optimal policy is to move the toilet seat alongside the closest boundary of the space. Consequently, the element with the type "IfcFlowTerminal" was designed to move while the other remains stationary.

For the initialization of the environment, the toilet was positioned in eight distinct regions within the bathroom of the house, as illustrated in Figure 6.2.

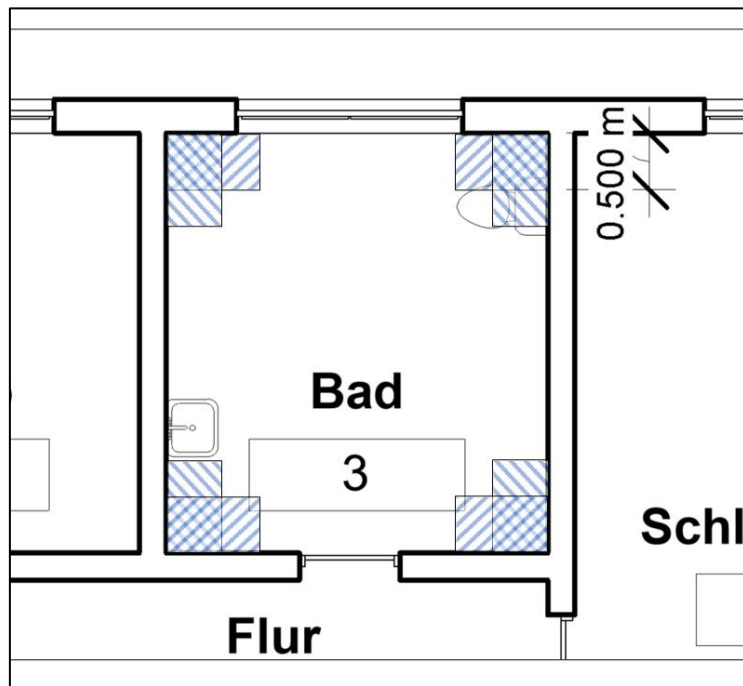


Figure 6.2: The initial placement regions of the toilet seat

The toilet is a pre-implemented plumbing fixture in Revit with a type of Toilet-Domestic-3D. It is imperative that a toilet be placed in each region, connected with one wall while with a distance smaller than 500 mm from another wall. The specific placement within the region is also randomly generated, thus enabling the agent to learn all eight different conflict scenarios with varying coordinates. In addition to the toilet, a Sink Vanity-Square 20" x 18", was added in the bathroom. This allowed for the toilet to potentially interact with the sink during the training process, thereby further enhancing the versatility of the training environment.

6.2.2 Results and Analysis

The algorithm commences its operation without any prior knowledge of the environment. Figure 6.3 provides an illustration of the execution of episode 1, delineating each step in a step-by-step manner. One unit in the scatterplot represents the movement distance of one action, i.e. 50 mm. The arrow indicates the direction of the action. As this is the inaugural instance of the algorithm processing each conflict, it attempts the available actions randomly and reaches the truncated state after 30 time steps.

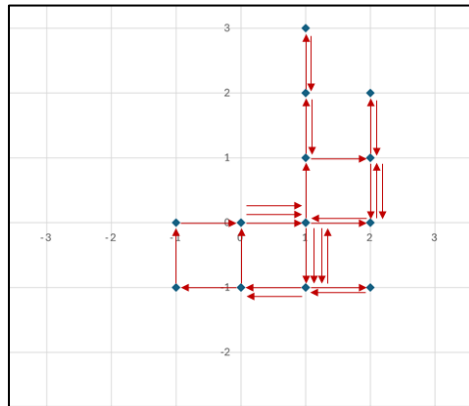


Figure 6.3: The illustration of the agent movement in the first training episode

Due to the training speed limit, the duration of the training of this use case is 10000 time steps. The training process itself takes approximately ten days. The most important evaluation diagram, which depicts the step and the reward, is presented in Figure 6.4. As the training time step increases, the rewards received by the agent exhibit oscillating behavior, indicating a lack of converge. An examination of the custom training process loggings reveals that the agent has learned to consistently move in a single direction, rather than randomly moving in all four directions. However, it has not yet learned to move in one specific direction based on the observation space.

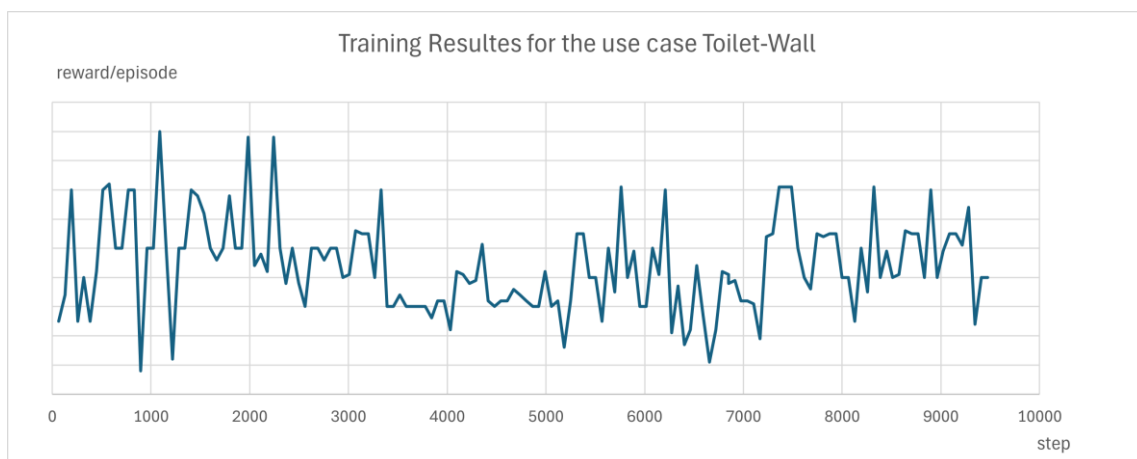


Figure 6.4: The training evaluation for the toilet-wall conflict

The underlying causes of the unsuccessful outcome can be attributed to two underlying factors. The primary issue lies in the insufficient number of training steps. The integration of Solibri guarantees the completeness and thoroughness of the detection process; however, it has also led to a decline in the training speed in comparison to training in a simulated environment. The limited speed of API calls and the relatively slow processing speed of the software are the primary contributing factors. One iteration of RL training can be a time-consuming process. In particular, as the number of training iterations increases, the time required for each round rises exponentially. The initial steps may require only a few seconds, whereas after several hundred steps, the time needed for the algorithm to run each iteration may increase to one minute and subsequently decrease in efficiency. Despite this observation, a method of training 1,024 steps at a time and then continuing the training manually was implemented to accelerate the process. However, the total time remained considerable.

The second reason pertains to the tuning of hyperparameters, which are of critical importance in the context of RL algorithms. While certain general guidelines exist, there is currently no efficient method for hyperparameter tuning. Rather, the process is largely based on trial and error. Despite the fact that several combinations of hyperparameters have been tested, the training effect has not been significantly enhanced, particularly given the constraints imposed by the number of training steps.

6.3 The Column-Window Conflict

In the field of architectural design and structural engineering, the placement of a column too close to a window is a common conflict. This issue arises when structural elements obstruct key architectural features, such as windows, which are crucial for natural lighting and ventilation. The commonality of this conflict stems from constraints in design, where the optimal placement of load-bearing columns coincides with window locations. This practice contravenes principles of visibility, occupant comfort and daylighting standard in DIN 5034:2021-08.

6.3.1 The Experiments

In practice, the type of conflict between the column and the window requires the input of architects and structural engineers. In order to determine which element should be moved or even deleted in the event of an extreme scenario, it is necessary to consider

a number of factors, such as whether the column is load-bearing and whether the window constitutes a component of a specific façade design. For the purposes of training, the process was simplified to entail the relocation of the column. The configuration is analogous to that of the previous subsection. A comprehensive illustration of the rulesets configuration in Solibri is provided in Figure 5.2 within subsection 5.1.1.

In accordance with the resolution assumption, the action space for this particular use case was also designed as a subset of the action space described in subsection 4.2. The available actions are as follows: The available actions are as follows: (1) positive x move, (2) negative x move, (3) positive y move, and (4) negative y move. The distance of each movement is 100 mm, given that the distance tolerance before the window is 1 m. Consequently, the agent is capable of resolving the issue up to a maximum of 10 time steps with the optimal policy, which entails moving the column in the opposite direction of the window. For each conflict, the maximum of 30 steps is constrained. Different from the previous conflict, the observation space for this use case is defined as "IfcColumn" and "IfcWindow", which represent their IFC classes.

To the reset of the training environment, the column was positioned in nine distinct areas distributed in front of each window in the house, as illustrated in Figure 6.5.

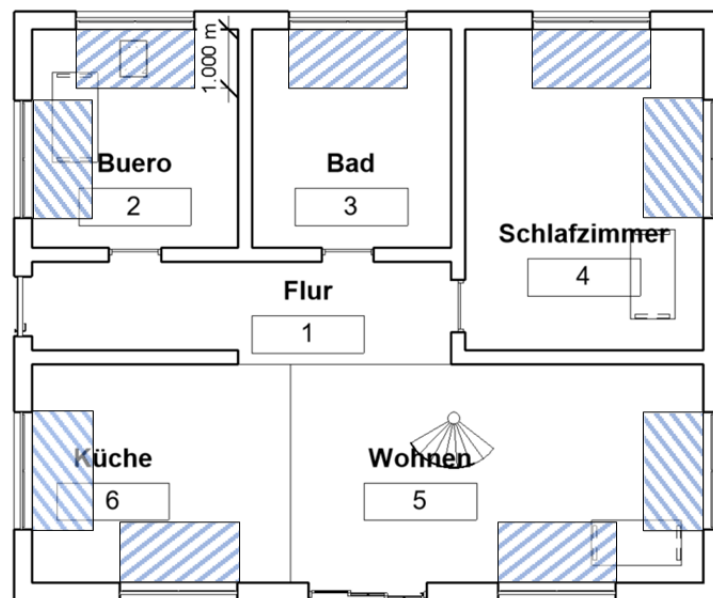


Figure 6.5: The initial placement area of the column

The length of the area is equivalent to the length of the window, while the width is 1 m. The column placement within the area was also randomly generated, ensuring that a column-window conflict was generated in the IFC file. The column that was added in

the original IFC file is a Concrete-Rectangular-Column 18 x 24 from Revit. In addition to the column, three tables: Furniture_System-Standing_Desk-Rectangular 60" x 30", were incorporated into the IFC model and situated in the office (Büero), living room (Wohnen), and bedroom (Schlafzimmer). This allowed for the column to potentially interact with the table during the training process, thereby emulating the actual scenario. In this manner, the agent is capable of acquiring the requisite coordinates for a multitude of potential conflict scenarios and the diversity of the training environment is guaranteed.

6.3.2 Results and Analysis

The algorithm initiates its operation without any knowing of the environment, in a manner analogous to that described in the preceding chapter. At the early stage of training, the algorithm randomly attempts the available operations and gradually improves. However, this use case presents the same issue as that observed in the first use case. The reward obtained by the agent has demonstrated a similar fluctuating pattern and has not reached a steady state as the training time step increases. As demonstrated in the preceding example, the agent has similarly acquired the capacity to move multiple steps in a single direction, as opposed to randomly in all four directions. However, in lieu of learning to move in a particular direction based on the observation space.

To address this issue, an alternative approach was taken, whereby extensive experimentation on hyperparameters tuning was conducted for this use case. This involved runs extending to 2048 time steps, with the aim of ascertaining the impact of hyperparameters. The main hyperparameters that were adjusted included:

- `ent_coef` (the entropy coefficient) to control the exploration,
- `learning_rate` (the learning rate) to facilitate the agent's adaptation to the environment,
- `batch_size` (the size of the batch to update the gradient) to stabilize the gradient update
- `n_steps` (the rollout buffer size) to change the time steps of observation before updating the policy

Moreover, given the discrete action space and the issue of insufficient data samples resulting from the limited training speed, training with the DQN algorithm was also attempted. As stated in Chapter 3, the DQN algorithm is an off-policy one, and thus

theoretically more suitable for situations where data is insufficient. Furthermore, for a discrete space, the DQN algorithm is a relatively simple choice. Figure 6.6 illustrates four of the training evaluation's diagrams.

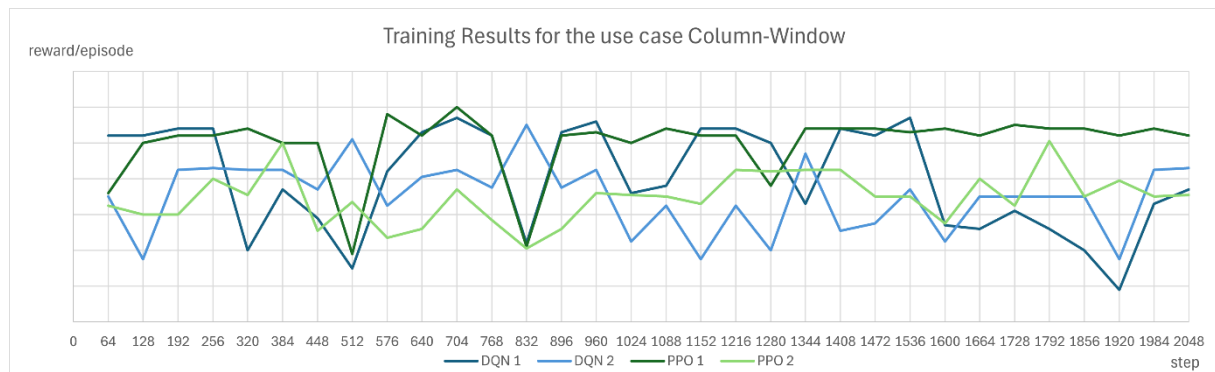


Figure 6.6: The training evaluation for the column-window conflict

The training results obtained with PPO are marginally superior, though the difference is not statistically significant. Another notable finding is that the training speed of the PPO is approximately 50% faster than that of DQN. This result further corroborates the hypothesis that the primary limitation for the training is the restricted number of training steps.

6.4 The Air Terminal-Door Conflict

A common conflict that arises in the integration of MEP systems with architectural design is the placement of air terminals, whether supply or return, in proximity to doorways. Positioning an air terminal too close to a door is generally discouraged, as the movement of the door can significantly disrupt the airflow, leading to inefficiencies in air distribution and potential discomfort for occupants. This design consideration is often guided by standards DIN EN 13779.

6.4.1 The Experiments

In consideration of the two preceding experiments, we proceed to further simplify the environment with the intention of attaining optimal training outcomes. Moreover, the movement of MEP objects is generally more challenging than that of other disciplines due to the fact that MEP elements, such as pipes, ducts and cables, are typically situated in specific systems and are all interconnected. The movement of a single element often necessitates the coordinated movement of numerous other related elements. Therefore, in this use case, we also seek to illustrate the versatility and capabilities of manipulating elements within IFC files.

The selected rulesets in Solibri are presented in Appendix A.2. The primary two rules are to check whether the air terminal is sufficiently distant from the frame of the doors and to check the intersection between building services and other surrounding components. The illustration of the conflict is shown in Figure 6.7. A simple mechanical supply-return air system was modeled in Revit to create conflicts. Four of the air terminals within the red circles with different directions were purposely positioned in front of the doors.

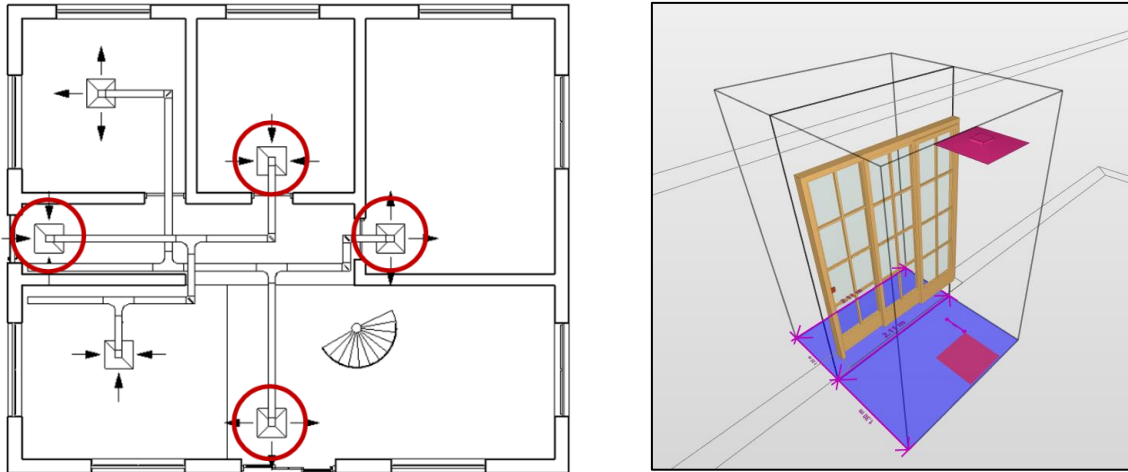


Figure 6.7: The illustration of the air terminal-door conflict

In this use case, it is assumed that the second floor is unoccupied and can be utilized for the positioning of the duct systems. In practice, this type of conflict is typically resolved by relocating the air terminal, particularly when there is sufficient space above the ceiling. Therefore, the action space comprises the same four actions required to move the air terminal in the x and y directions. To accelerate the training process, two modifications have been implemented:

- The distance of each movement was set to 200 mm, thus enabling the resolution of conflicts in a more efficient manner. Under the optimal policy, one conflict can be resolved in five steps.
- A knowledge-based restriction was introduced into the action space, whereby an action that does not result in an increase or decrease in the distance between the door and the air terminal will not be performed. In this case, the agent is given a penalty of $r = -0.5$ directly, and advance to the next step.

Different from the other two use cases, in the case of air terminal-door conflict, the requisite movement is contingent upon the corresponding movement of the related ducts and fittings. The utilization of the “port” class and the relations defined in the IFC

model enables the systematic tracing of related elements. Ports serve as pivotal connection points for defining the flow relationships between different building services components. It facilitates the seamless interconnection of elements including ducts, fittings, and air terminals. In this use case, the diffuser is connected to an *inport* with the relation type of *IfcRelNests*, which indicates that the port is a subcomponent of the air terminal. In this context, the *IfcRelConnectsPorts* relationship is employed to define the connection between the port on the air terminal and the port on the duct, specifying the direction of the airflow. These relationships facilitate accurate modeling and the hierarchical structure of the IFC file, thereby enabling the retrieval of all related elements' GUID in this use case: the short duct, the fitting and the long duct. In addition to the air terminal, the first two connected elements should also be relocated, and the long duct should be shortened or lengthened accordingly.

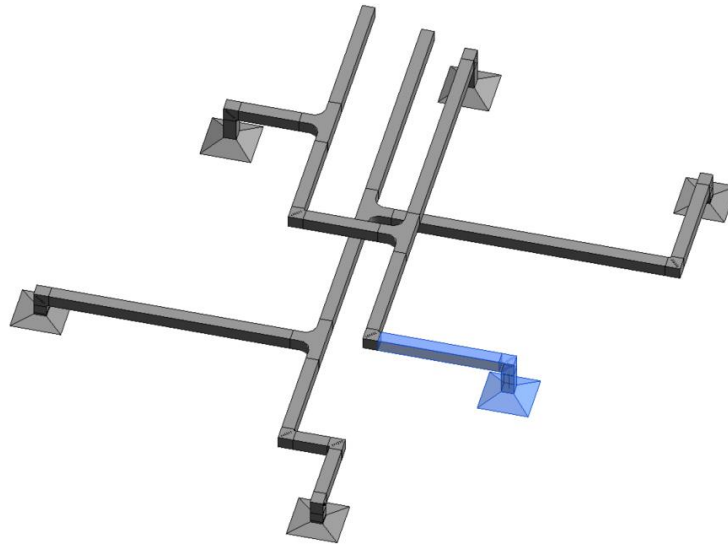


Figure 6.8: The illustration of the agent for the air terminal-door conflict

In total, four conflicts must be resolved with a total of 20 steps required to achieve this under the optimal policy. The maximum step limit is set at three times the number of necessary steps, as previously stated, equating to 60 steps. With regard to the action space, it is assumed that the agent performs invalid moves half of the time; therefore, the maximum step limit is finally set to 120 steps. The element type parameter is set to "IfcAirTerminal" and "IfcDoor". The same IFC model was used to reset the RL environment each time.

6.4.2 Results and analysis

The training speed of this experiment is evidently more rapid than that of the former two experiments, primarily due to the presence of two invalid actions. In particular, during the initial stages of training, the agent would consistently select the invalid action until reaching the truncation state of this episode. As illustrated in Figure 6.8, following a significant reduction in complexity, the agent demonstrated the capacity to effectively address the four specific conflicts. After training, in ten subsequent tests employing the trained RL model, the RL model exhibited an average of 32 steps to successfully resolve the four conflicts.

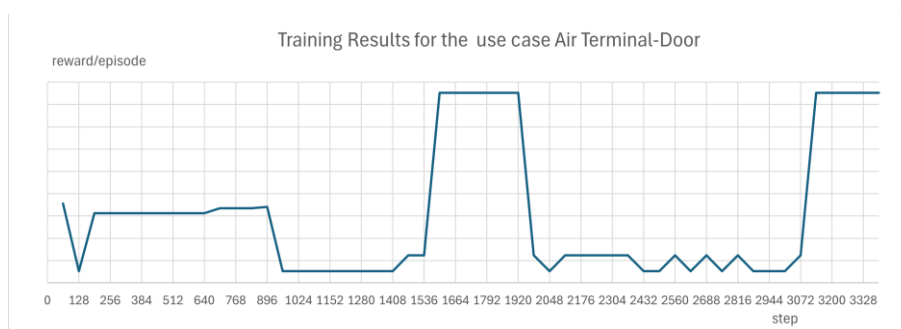


Figure 6.9: The training evaluation for the air terminal-door conflict

For a scenario with four fixed conflicts, the agent needs around 1600 training steps to get a good result. It is reasonable to assume, that for the other two experiments with much more variations of conflicts, the necessary training steps would increase exponentially.

7 Conclusions and Future Works

7.1 Conclusions

The objective of this study was to investigate the approach of automating the conflict resolution process through the implementation of a PPO-based RL algorithm within an integrated BIM environment utilizing an IFC model checker. The main contributions are as follows:

1. The integration of the IFC model checker, Solibri, with the Python-programmed formulated RL environment.
2. The proposal of the PPO-based RL algorithm to interact with the environment.
3. The definition of different conflict types and checking rules that are common in the AEC industry to test the feasibility of the proposed framework.
4. The training of the RL agent in three different environments separately to resolve the conflicts.

A conflict resolution RL algorithm based on PPO techniques was presented, which does not require initial input data to identify the optimal sequence of available resolution actions for a list of conflicts. The Solibri software was employed to detect and export the data pertaining to the conflicts within the IFC model, including the components that are in conflict, as well as the severity of the conflicts. Based on this, the agent is able to observe the integrated environment. The parameters selected for observation include the number of conflicts, the severity of the conflicts, the IFC type of the components, the rotations of the components, and the vertices of the components. Subsequently, the agent selects the optimal action in accordance with the current policy, which is based on the PPO algorithm. The action is executed directly within the IFC file, which is integrated as a component of the RL environment. A bespoke reward function has been devised for the agent in conjunction with the model checker, enabling the provision of the feedback. The reward function comprises three key changes: the number of conflicts, the number of newly created conflicts and the severity of the conflicts. Its applicability is evaluated through three experiments, which consider conflicts in architectural sub-aspects, between architectural and structural elements, and

between architectural and MEP elements. The experiment was successful in a simplified environment. The specific inheritance structure of the IFC model allows for a wide range of applications in different scenarios and the effective manipulation of different actual IFC elements.

7.2 Limitations

As pioneering research into the use of RL for BIM conflict resolution automation, the results of the experiments applying the proposed approach did not meet expectations. The following section will discuss the potential explanations for these outcomes and the associated limitations.

The inadequate training step is the primary factor contributing to the unsuccessful training outcomes, which is attributable to the sluggish training speed of the environment. Model-free RL algorithms are relatively sample inefficient. They require a substantial number of samples, often millions of interactions to achieve something meaningful, which is the key reason why most of the successes in RL were achieved on games or in simulation only (StableBaselines3, 2024). However, the integration of Solibri resulted in a discernible reduction in training speed during the training process. It was observed that initially, the training process exhibited a significantly faster runtime compared to subsequent phases. Besides, the CPU usage is relatively low, generally below 10%, while the memory usage is considerably higher, at 80-90%. It seems reasonable to posit that the accumulation of data and parameters over time may have increased memory usage and processing overhead, thereby further slowing down the training process. Furthermore, the iterative nature of RL, in conjunction with the necessity to frequently access, update and check the IFC model via Solibri, may have introduced delays due to the extensive I/O operations involved. The Solibri REST API calls are made over a network, and the condition of the network may change over training process, resulting in increased latency. These factors collectively contribute to the observed reduction in training speed over time. However, throughout the training process, the computer with the larger RAM capacity tends to complete the training process at a slower rate than the computer with 16 GB of memory. One potential explanation for this discrepancy is that larger memory systems are shared among multiple users, and the lack of full administrative privileges on the computer with the larger RAM capacity may have restricted certain optimizations and configurations that could enhance training efficiency. Additionally, the software updates and configurations on the computer

with the 32 GB RAM may have been limited, which could also have contributed to the observed slowness in training speed.

A further limitation to the generalizability the framework is that the model is trained separately in three scenarios due to the lengthy training times involved. From a practical standpoint, it would be advantageous to merge the model into a single one, as this would enable the model to resolve all conflicts. This is a theoretically feasible approach, given that the three models share a common framework. However, to enhance the efficiency of the agent's learning process and facilitate the optimal policy, certain necessary adjustments were made to the distance of each movement, the available actions, and the number of IFC types. If one model is trained for the three use cases, with each move being 10mm and invalid z-axis moves or rotations included, the number of steps required to resolve the conflict would increase exponentially. It is therefore evident that the training process will be even more time-consuming.

Another limitation is associated with its applicability across different models. During the training process, only a standard one-family house IFC model with relatively simple geometry was utilized. Despite the random positioning of the conflicting components at the initialization of the RL environment and the addition of other elements around to ensure the diversity and complexity of the conflicts in a certain degree, the application of the trained model to larger and more complex models still requires further testing and evaluation even if the training results of the three use cases are successful.

The experiments conducted to tune hyperparameter tuning were insufficient. The selection of hyperparameters in RL has a significant impact on the rate of convergence, the stability of the learning process, and the overall success of the learning task. Properly tuned hyperparameters has the potential to markedly improve the efficiency of the learning process and reduce training time. Nevertheless, Tuning RL hyperparameter does not have clear and sufficient scientific principles to work with (Li, 2018) and the process of tuning these hyperparameters is notoriously challenging due to the high dimensionality of the hyperparameter space of PPO and the stochastic nature of RL environments. As a result, the identification of the optimal set of hyperparameters remains a time-consuming and computationally expensive task in RL.

7.3 Future Works

In response to the discussions and limitations identified in the previous subsection, our future work will focus on improving the training speed, expanding the framework to encompass a broader range of use cases, and applying the trained model to a variety of more complex situations.

The primary objective is to investigate the potential for enhancing the integration of Solibri in order to facilitate a more expeditious training process, which serves as a fundamental basis for the subsequent improvements. Solibri provides a Java API, which is primarily utilized for the purpose of customizing rulesets. However, the potential for integrating its Java API and the Python RL environment directly could prove an effective method for accelerating the training process. This eliminates the necessity for REST API calls, thereby accelerating the training process. Moreover, it may be feasible to examine a specific region of the IFC model surrounding the conflicting components, as the significant portion of the model remains unchanged throughout the training process. By reducing the number of components that require examination each time, the training process could be made to proceed more rapidly. Furthermore, the use of parallel environments could facilitate more efficient training and utilization of the CPU. As Solibri permits the inclusion of multiple models within a single SMC file, it may be feasible to envisage a scenario in which disparate agents can be controlled across distinct models simultaneously. It may also be beneficial to investigate the memory allocation of the computer in order to achieve a more balanced usage of CPU and memory. This could entail the fine-tuning of the system's resource allocation policies or the implementation of more efficient algorithms for the handling of large datasets in memory during training. Furthermore, enhancing training efficiency may be achieved by utilizing more powerful hardware.

The exploration of unify the separately trained RL into one model also represents a significant avenue for future research. Although the current proposed framework is theoretically feasible, it requires meticulous calibration of the agent's learning parameters and a more precise framework. Further refinement of the reward, action, observation module could prove beneficial. For example, breaking different rules in Solibri could be subject to disparate penalties, which would be determined in accordance with the applicable building regulations. Alternatively, the distance of the movement could not be defined in accordance with the element's IFC type and the broken rule, rather

than a fixed value of 10mm. A more comprehensive RL system could prevent exponential growth in the required training steps. Research could focus on refining these parameters to maintain a balance between generalizability and training efficiency. Further exploration of alternative RL algorithms, such as DQN, could also provide insights into potentially more suitable methods for the specific problem at hand.

Moreover, future studies should include the testing and evaluation of the RL model across a range of IFC models, encompassing varying degrees of conflicts complexity and structural diversity, rather than relying exclusively on the standard one-family house IFC model employed in the present study. The complexity of the model should be incrementally augmented, commencing with the successful conflict resolution of relatively simple environments and subsequently progressing to the testing or further training in somewhat more complex models. The implementation of a gradual progression reduces the amount of unnecessary training time. This could ensure the robustness and applicability of the model in a broader range of practical settings. It would also be beneficial to investigate how to increase the diversity of the initialized conflicts in different IFC models. In the current experiments, the conflicted components' type and dimension are the same, only the placement and orientation are randomly generated. Implementing advanced techniques for automated generation of conflict scenarios, such as GAN, could further enrich the diversity of training data.

Finally, with regard to hyperparameter optimization, future work should incorporate the use of automated hyperparameter optimization tools, such as the Bayesian optimization strategy or the Optuna optimization framework, for the systematic tuning of the parameters of the PPO algorithm. A comprehensive analysis of the training parameters throughout the training process should contribute to more effective hyperparameter tuning.

References

- Bellman, R. (1956). DYNAMIC PROGRAMMING AND LAGRANGE MULTIPLIERS. *Proceedings of the National Academy of Sciences*, 42(10), 767–769. <https://doi.org/10.1073/pnas.42.10.767>
- Borrmann, A., Beetz, J., Koch, C., Liebich, T., & Muhic, S. (2018). Industry Foundation Classes: A Standardized Data Model for the Vendor-Neutral Exchange of Digital Building Models. In A. Borrmann, M. König, C. Koch, & J. Beetz (Eds.), *Building Information Modeling* (pp. 81–126). Springer International Publishing. https://doi.org/10.1007/978-3-319-92862-3_5
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym* (arXiv:1606.01540). arXiv. <http://arxiv.org/abs/1606.01540>
- Building and Construction Authority (2013). *Singapore BIM Guide Version 2*. https://www.corenet.gov.sg/media/586132/Singapore-BIM-Guide_V2.pdf
- buildingsmart (2024). *Annex D (informative) diagrams*. Annex D (informative) Diagrams IFC4.3.2.0 Documentation. https://standards.buildingsmart.org/IFC/RELEASE/IFC4_3/HTML/annex_d/lfcWindow.html#Figure-D.A
- BuildingSMART technical (2024). *Industry Foundation Classes (IFC)* <https://technical.buildingsmart.org/>
- Charehzehi, A., Chai, C., Md Yusof, A., Chong, H.-Y., & Loo, S. C. (2017). Building information modeling in construction conflict management. *International Journal of Engineering Business Management*, 9, 184797901774625. <https://doi.org/10.1177/1847979017746257>

- Chen, H.-M., & Hou, C.-C. (2014). Asynchronous online collaboration in BIM generation using hybrid client-server and P2P network. *Automation in Construction*, 45, 72–85. <https://doi.org/10.1016/j.autcon.2014.05.007>
- Christopher Watkins. (1989, May). *Learning from delayed rewards*. Cambridge University of Cambridge.
- García, C. E., Prett, D. M., & Morari, M. (1989). Model predictive control: Theory and practice—A survey. *Automatica*, 25(3), 335–348. [https://doi.org/10.1016/0005-1098\(89\)90002-2](https://doi.org/10.1016/0005-1098(89)90002-2)
- Gijzen, S., Hartmann, T., Veenvliet, K. T., Hendriks, H., & Buursema, N. (2010). *Organizing 3D Building Information Models with the help of Work Breakdown Structures to improve the Clash Detection process*.
- Harode, A., & Thabet, W. (2021). *Investigation of Machine Learning for Clash Resolution Automation*. 228–218. <https://doi.org/10.29007/n223>
- Harode, A., Thabet, W., & Gao, X. (2022). An Integrated Supervised Reinforcement Machine Learning Approach for Automated Clash Resolution. *Construction Research Congress 2022*, 679–688. <https://doi.org/10.1061/9780784483961.071>
- Harode, A., Thabet, W., & Gao, X. (2024). Developing a Machine-Learning Model to Predict Clash Resolution Options. *Journal of Computing in Civil Engineering*, 38(2), 04024005. <https://doi.org/10.1061/JCCEE5.CPENG-5548>
- Hasannejad, A., Shirzadi Javid, A. A., & Bitaraf, I. (2023). BIM-based Clash Resolution Process Using Fuzzy AHP Methods. *Iranian Journal of Science and Technology, Transactions of Civil Engineering*. <https://doi.org/10.1007/s40996-023-01238-z>

- Hsu, H.-C., Chang, S., Chen, C.-C., & Wu, I.-C. (2020). Knowledge-based system for resolving design clashes in building information models. *Automation in Construction*, 110, 103001. <https://doi.org/10.1016/j.autcon.2019.103001>
- Huang, Y.-H., & Lin, W. Y. (2019, May 24). *Automatic Classification of Design Conflicts Using Rule-based Reasoning and Machine Learning An Example of Structural Clashes Against the MEP Model*. 36th International Symposium on Automation and Robotics in Construction, Banff, AB, Canada. <https://doi.org/10.22260/ISARC2019/0044>
- IBM (2024). *What is an API (application programming interface)?*. https://www.ibm.com/topics/api?mhsrc=ibmsearch_a&mhq=API
- IfcOpenShell (2024), *IfcOpenshell-Python*. <https://docs.ifcopenshell.org/ifcopenshell-python.html>
- Isikdag, U., & Underwood, J. (2010). Two design patterns for facilitating Building Information Model-based synchronous collaboration. *Automation in Construction*, 19(5), 544–553. <https://doi.org/10.1016/j.autcon.2009.11.006>
- Jeong, J., & Jo, H. (2021). Deep reinforcement learning for automated design of reinforced concrete structures. *Computer-Aided Civil and Infrastructure Engineering*, 36(12), 1508–1529. <https://doi.org/10.1111/mice.12773>
- Kubicki, S., Guerriero, A., Schwartz, L., Daher, E., & Idris, B. (2019). Assessment of synchronous interactive devices for BIM project coordination: Prospective ergonomics approach. *Automation in Construction*, 101, 160–178. <https://doi.org/10.1016/j.autcon.2018.12.009>
- Lange, S., Riedmiller, M., & Voigtlander, A. (2012). Autonomous reinforcement learning on raw visual input data in a real world application. *The 2012 International*

- Joint Conference on Neural Networks (IJCNN)*, 1–8.
<https://doi.org/10.1109/IJCNN.2012.6252823>
- Lehmann, M. (2024). *The Definitive Guide to Policy Gradients in Deep Reinforcement Learning: Theory, Algorithms and Implementations* (arXiv:2401.13662). arXiv.
<http://arxiv.org/abs/2401.13662>
- Li, Y. (2018). *Deep Reinforcement Learning: An Overview* (arXiv:1701.07274). arXiv.
<http://arxiv.org/abs/1701.07274>
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2019). *Continuous control with deep reinforcement learning* (arXiv:1509.02971). arXiv. <http://arxiv.org/abs/1509.02971>
- Liu, J., Liu, P., Feng, L., Wu, W., & Lan, H. (2019, May 24). *Automated Clash Resolution of Rebar Design in RC Joints using Multi-Agent Reinforcement Learning and BIM*. 36th International Symposium on Automation and Robotics in Construction, Banff, AB, Canada. <https://doi.org/10.22260/ISARC2019/0123>
- Liu, J., Liu, P., Feng, L., Wu, W., Li, D., & Chen, Y. F. (2020). Automated clash resolution for reinforcement steel design in concrete frames via Q-learning and Building Information Modeling. *Automation in Construction*, 112, 103062. <https://doi.org/10.1016/j.autcon.2019.103062>
- Liu, P., Qi, H., Liu, J., Feng, L., Li, D., & Guo, J. (2023). Automated clash resolution for reinforcement steel design in precast concrete wall panels via generative adversarial network and reinforcement learning. *Advanced Engineering Informatics*, 58, 102131. <https://doi.org/10.1016/j.aei.2023.102131>

- Liu, X., Zhao, J., Yu, Y., & Ji, Y. (2024). BIM-based multi-objective optimization of clash resolution: A NSGA-II approach. *Journal of Building Engineering*, 89, 109228. <https://doi.org/10.1016/j.jobbe.2024.109228>
- Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of machine learning* (Second edition). The MIT Press.s
- Postman API Platform. (2024). *What is an API? A beginner's guide to apis*: <https://www.postman.com/what-is-an-api/#how-do-apis-work>
- Radke, A. M., Wallmark, T., & Tseng, M. M. (2009). An automated approach for identification and resolution of spatial clashes in building design. *2009 IEEE International Conference on Industrial Engineering and Engineering Management*, 2084–2088. <https://doi.org/10.1109/IEEM.2009.5373167>
- Reinforcement Learning and Its Relationship to Supervised Learning. (2009). In J. Si, A. G. Barto, W. B. Powell, & D. Wunsch, *Handbook of Learning and Approximate Dynamic Programming*. IEEE. <https://doi.org/10.1109/9780470544785.ch2>
- Rummery, G. A., & Niranjan, M. (1994). *ON-LINE Q-LEARNING USING CONNECTIONIST SYSTEMS*.
- Sacks, R., Eastman, C., Lee, G., & Teicholz, P. (2018). *BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers* (1st ed.). Wiley. <https://doi.org/10.1002/9781119287568>
- Scherer, R. J. (2007). *PRODUCT MODEL BASED COLLABORATION*.
- Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2017). *Trust Region Policy Optimization* (arXiv:1502.05477). arXiv. <http://arxiv.org/abs/1502.05477>

- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms* (arXiv:1707.06347). arXiv. <http://arxiv.org/abs/1707.06347>
- Sharbaf, M., Zamani, B., & Sunyé, G. (2022). Automatic resolution of model merging conflicts using quality-based reinforcement learning. *Journal of Computer Languages*, 71, 101123. <https://doi.org/10.1016/j.cola.2022.101123>
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484–489. <https://doi.org/10.1038/nature16961>
- Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). *Deterministic Policy Gradient Algorithms*.
- Solibri office (2024). *the core product for model checking and collaboration* <https://www.solibri.com/solibri-office>
- Stable Baselines3 *Reinforcement Learning Tips and Tricks*. https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html
- Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.
- Tizani, W. (2010). *Computing in civil and building engineering: Proceedings of the 13th International Conference on Computing in Civil and Building Engineering and the 17th International EG-ICE Workshop on Intelligent Computing in Engineering, 30 June - 2 July, Nottingham, UK*. Nottingham University Press.









- Wang, L., & Leite, F. (2016). Formalized knowledge representation for spatial conflict coordination of mechanical, electrical and plumbing (MEP) systems in new building projects. *Automation in Construction*, 64, 20–26. <https://doi.org/10.1016/j.autcon.2015.12.020>
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4), 229–256. <https://doi.org/10.1007/BF00992696>
- Xie, X., Zhou, J., Fu, X., Zhang, R., Zhu, H., & Bao, Q. (2022). Automated Rule Checking for MEP Systems Based on BIM and KBMS. *Buildings*, 12(7), 934. <https://doi.org/10.3390/buildings12070934>
- Yang, C., Zheng, Z., & Lin, J.-R. (2023). *Automatic Design Method of Building Pipeline Layout Based on Deep Reinforcement Learning*.

Appendix A

A.1 Predefined Rulesets in Solibri for Toilet-Wall Conflict

Name	Support Tag
▼ Rulesets Open in Solibri	
▼ § #209 Free Floor Space	
§ 403.5 Clear Width	SOL/209/2.0
§ 804.2.1 Pass Through Kitchen	SOL/209/2.0
§ 11.3 Turning space for 90° turn of a wheelchair in corridors	SOL/209/2.0
§ 11.4 Circulation space for 180° wheelchair turn	SOL/209/2.0
§ 19.3 Space to manoeuvre	SOL/209/2.0
§ 26.15 & 26.17 Shower & Bathrooms	SOL/209/2.0
▼ § #222 Component Distance	
§ Distance Between Toilet Seat and Wall	SOL/222/4.2
▶ § BIM Validation - Architectural	
▶ § General Space Check	
▼ § Intersections Between Architectural Components	
▶ § Intersections - Same Kind of Components	
▶ § Intersections - Different Kind of Components	
▼ § Intersections of Furniture and Other Objects	
§ Object Intersections	SOL/1/5.0
§ Doors/Windows and Objects	SOL/1/5.0
§ Objects and Other Components	SOL/1/5.0

A.2 Predefined Rulesets in Solibri for Air Terminal-Door Conflict

Name	Support Tag
▼  Rulesets Open in Solibri	
▶  BIM Validation - Architectural	
▶  General Space Check	
▶  Intersections Between Architectural Components	
▼  MEP models and Architectural model	
▼  Building Services and Architectual Components	
§ Building Services and Doors and Windows	SOL/1/5.0
§ Building Services and Beams and Columns (Insulations Not Included)	SOL/1/5.0
§ Building Services and Other Construction Components (Insulations Not Included)	SOL/1/5.0
§ Building Services and Furniture and Other Objects	SOL/1/5.0
§ Insulations and Beams and Columns	SOL/1/5.0
§ Insulations and Other Construction Components	SOL/1/5.0
▼  Distance between Components	
§ Distance Between Columns/Beams and MEP components	SOL/222/4.2
§ Distance Between Doors and MEP components	SOL/226/3.1
§ Distance Between Windows and MEP components	SOL/226/3.1
▶  Spaces and MEP	

Declaration

I hereby affirm that I have independently written this Master's thesis submitted by me and have not used any sources or aids other than those indicated.

I also confirm that this thesis has not been the basis of any other examination procedure.

During the preparation of this thesis, I used OpenAI's tool, ChatGPT and DeepL's tool DeepL Write, in order to proofread the manuscript. After using the tools, I reviewed and edited the content as needed and take full responsibility for the content of the thesis.

München, 29. August 2024

Yuye Jiang

Name Surname