Politécnico
Castelo Branco
Escola Superior
de Tecnologia

ECSMoS: An ECS-based Pedestrian Mobility Simulator

Rafael Souza Cotrim

Orientadores

Alexandre José Pereira Duro da Fonte

João Manuel Leitão Pires Caldeira

Dissertação apresentada à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática - Área de Especialização em Desenvolvimento de Software e Sistemas Interativos, realizada sob a orientação científica do Professor Adjunto Alexandre José Pereira Duro da Fonte e Professor Adjunto João Manuel Leitão Pires Caldeira, do Instituto Politécnico de Castelo Branco.

Outubro 2025

Composição do ~~Juri~~Júri

Presidente do júri

Grau académico, nome do presidente do júri

Vogais

Grau académico, nome do presidente do júri

Categoria profissional e o nome da Instituição

Grau académico, nome do presidente do júri

Categoria profissional e o nome da Instituição

Grau académico, nome do presidente do júri

Categoria profissional e o nome da Instituição

Resumo

~~Este resumo não deverá ter uma extensão superior a 500 palavras em português, inglês ou espanhol~~The study and simulation of the movement of pedestrians, often referred to as pedestrian dynamics or crowd dynamics, is of fundamental importance in today's world. It is used in many kinds of applications, including the optimization of spaces where large amounts of people pass through and the prevention of crowd crushes, stampedes, tramplings, and other similar events.

Currently, the research and development of new models that mimic the behavior of people walking is restricted by the available solutions. Existing pedestrian simulators oftentimes have limitations imposed by their architectures, leading researchers to create their own simulators, which slows down research and limits comparability of results. This dissertation evaluates the utilization of the Entity Component Systems architecture to improve the flexibility of such software. Additionally, it introduces a new pedestrian simulation framework, ECSMoS, which focuses on solving the flexibility limitations.

To arrive at this result, first it was necessary to study current models. Then, a survey of the available simulators was performed. The strengths and weaknesses of each were noted for later use in the definition of the objectives and testing scenarios of the new simulator. Then, an introduction to the Entity Component Systems Architecture was given and how it may solve the flexibility limitations of current simulators was explained. With these definitions in place, this work defined the requirements for the ECSMoS simulator, explained its structure and compared it to other simulators.

Results show that, while more complex than the other simulators, ECSMoS is considerably more flexible, especially in situations where the environment is in constant flux. In terms of performance, it was shown that ECSMoS is just as or more performant than other simulators in the evaluated scenarios.

Palavras chave

~~Cinco palavras chave no máximo.~~ Pedestrain Dynamics, Simulation, ECS, Comparative Analysis

Abstract

~~Este resumo não deverá ter uma extensão superior a 500 palavras, por regra apresenta-se em inglês. Nos casos em que o documento seja escrito em língua inglesa ou espanhola, o abstract apresenta-se em português.~~ O estudo e a simulação do movimento de pedestres, chamado dinâmica de pedestres ou dinâmica de multidões, é de fundamental importância no mundo atual. É utilizado em diversos tipos de aplicações, incluindo a otimização de espaços por onde passam grandes quantidades de pessoas e a prevenção de esmagamentos, tumultos, debandadas e outros eventos semelhantes.

Atualmente, a pesquisa e o desenvolvimento de novos modelos que imitam o comportamento de pessoas andando está restrita pelas soluções disponíveis. Simuladores de pedestres existentes frequentemente possuem limitações impostas por suas arquiteturas, levando pesquisadores a criarem os seus próprios simuladores, o que desacelera a pesquisa e limita a comparabilidade dos resultados. Esta dissertação avalia a utilização da arquitetura de Sistema de Componente e Entidade para melhorar a flexibilidade desse tipo de software. Além disso, introduz um novo simulador de pedestres, o ECSMoS, que foca em resolver as limitações de flexibilidade.

Para alcançar esse resultado, foi necessário, primeiramente, estudar os modelos atuais. Em seguida, realizou-se um levantamento dos simuladores disponíveis. Os pontos fortes e fracos de cada um foram avaliados para uso posterior na definição dos objetivos e dos cenários de teste do novo simulador. Depois, foi feita uma introdução à arquitetura de Sistema de Componente e Entidade e explicado como ela pode resolver as limitações de flexibilidade dos simuladores atuais. Com essas definições estabelecidas, este trabalho definiu os requisitos para o simulador ECSMoS, explicou a sua estrutura e o comparou com outros simuladores.

Os resultados mostram que, embora mais complexo do que os outros simuladores, o ECSMoS é consideravelmente mais flexível, especialmente em situações onde o ambiente está em constante mudança. Em termos de desempenho, foi demonstrado que o ECSMoS tem desempenho tão bom quanto ou ainda melhor do que os outros simuladores nos cenários avaliados.

Keywords

~~Cinco palavras chave no máximo.~~ Dinâmica de pedestres, Simulação, ECS, Estudo Comparativo

List of Abbreviations

CSV Comma-separated values.

EC Entity-Component.

ECS Entity Component Systems.

ECSMoS Entity Component Systems Mobility Simulator.

FDS Fire Dynamics Simulator.

MVC Model View Controller.

OOP Object-Oriented programming.

SFM Social Forces Model.

SUMO Simulation of Urban Mobility.

UI User Interface.

General Index

Figure Index

Table Index

1. Introduction

Pedestrian dynamics is an area of study focused on understanding the movement of pedestrians, ~~which often happens~~ both individually and as crowds. ~~Such studies can take the form of an analysis of~~ Research in this area often involves analyzing data collected from ~~studying~~ real-world observations of the movements of people ~~in the real world [1], proposition and evaluation of~~ [1], developing and evaluating methods for modeling crowd behavior [2], and predicting movement patterns ~~in certain~~ within specific spaces [3], among ~~others. They may also take into account other factors, such as vehicles [4] and changes to the environment [5].~~ other topics. Studies on pedestrian dynamics may also consider interactions with vehicles [4] or the effects of environmental changes [5].

As such, pedestrian dynamics are of practical use when creating spaces meant to be utilized by people. ~~A~~ For instance, a better understanding of how people move through a structure may aid during the design phase of a building, allowing for better planning of ~~fire escape~~ evacuation routes [6]. Similarly, not taking into account how people will behave may turn concerts or other large ~~events~~ gatherings into deadly crowd crushes ~~and~~ or increase the number of trampling incidents [7]. Finally, even ~~when there is little risk of loss of life, they may still be useful for increasing the throughput of infrastructure~~in less risky situations, knowledge of pedestrian flow can improve the efficiency of infrastructure, such as train stations, airports, and others [8].

One of the ~~best tools from this area of study comes in the form of simulation models~~most valuable tools in the study of pedestrian dynamics is the use of simulation models [9]. Computer simulations ~~allow us to check how pedestrians will behave in a certain environment without having to spend~~ make it possible to evaluate how pedestrians might behave in specific environments without the need to invest time and resources ~~on the construction and evaluations of scale modelsin the real world. This reduces costs and promotes fast iterative designs that may better align with the requirements of spaces. It also bypasses ethical considerations that arise when attempting to replicate crowd crushes and other deadly situationsfor research. Study about these can happen via the~~ in constructing and testing physical scale models. This approach reduces costs, enables rapid iterative design, and helps ensure that spaces are better aligned with their intended use. Simulations also circumvent the ethical challenges associated with studying dangerous scenarios such as crowd crushes or other life-threatening situations. Although some insights can be obtained from records of past ~~events, but these are limited as these rarely happen under circumstances where data can be effectively collected. For these scenarios, simulations are some of the only ways to gather more profound insight~~incidents, such events rarely occur under conditions that allow for systematic data collection. Consequently, simulations remain one of the few viable methods for gaining a more in-depth understanding of these complex and hazardous phenomena.

~~There are many models that have been made for simulating the flow of pedestrians in an environment~~A wide variety of models have been developed to simulate pedestrian flow within different environments [6]. However, designing and implementing models ~~capable of reproducing phenomena seen in the real world is complicated~~that accurately reproduce real-world behaviors remains a challenging task. Pedestrian dynamics is ~~an inherently interdisciplinaryscience [10] due to its object of study. It relies on concepts from areas as diverse as~~ inherently interdisciplinary, drawing upon concepts from fields such as physics, engineering, psychology, computer science, and sociology [10]. Simpler models may take into account only the physical part of crowd movement and ignore all else, while others deal with the effects of having people with different ages [11], ~~disabilities [12]or~~ physical abilities [12], and even states of mind, such as calm ~~and~~ or panic [13].

When these models are implemented, it is often done on top of an existing simulator or framework. These allow model authors to focus on the most relevant parts of their research while other tasks are handled by code already written and validated by others. Nevertheless, building a model on top of these simulators also comes with certain disadvantages. Their architecture imposes restrictions on how the model can operate, meaning that certain simulators may not be compatible with a model ~~because~~ given it breaks one or more of the assumptions made when the simulator was being designed. Similarly, the architecture and technologies used by the simulator have an effect on the performance of the simulation. While less impactful than full incompatibility between model and simulator, low performance can slow model development and usage. Larger or more complicated scenarios may take longer than desired to be evaluated or require additional hardware.

While there are many ways of mitigating such problems, using architectures found in other areas, such as Entity Component Systems (ECS), may offer some benefits ~~.~~ [14]. In effect, ECS is a software architecture sometimes used in game development due to its much stricter flexibility and performance requirements than standard simulators. Developers expect such frameworks to be able to grow as the needs of the project expand and to handle thousands of entities updating multiple times a second. Not only that, but, as this work will show, these benefits can be ~~used~~ experienced in many scenarios, including the study of crowds. There has been some previous work in using ECS in other areas of research, namely in the area of co-simulations [15] and simulating radars [16]. Despite this, ECS is still very uncommon in the scientific world.

With this context in mind, this work proposes a new simulation framework for pedestrian dynamics: ECSMoS. This simulator is ~~based on~~ built upon the ECS architecture and ~~has a focus on being as flexible as possible for model authors while maintaining a~~ is designed to offer maximum flexibility for model developers while maintaining high performance. ~~Its main objective is to determine~~ The main objective of this research is to evaluate the viability, benefits, and ~~drawbacks of~~ limitations of using such an architecture in the field of pedestrian ~~simulations. This is done by implementing~~ simulation. To this end, the core of a novel simulation framework ~~and comparing it~~ was implemented and compared to existing simulators in various scenarios.

~~The rest of this work~~

1.1 Planning

The work required to complete this dissertation was divided into the tasks listed below. Tasks 1-4 were carried out sequentially, while Task 5 was performed in parallel with the others.

1. Study the current state of pedestrian simulations, with a focus on the models used.
2. Research the following topics:
   (a) Existing simulation frameworks used in pedestrian dynamics.
   (b) Tools and libraries currently used for implementing the ECS architecture.
3. Propose and implement a novel simulation framework.
4. Perform a comparative analysis of the proposed framework and existing ones in terms of performance and flexibility.
5. Write and edit the final version of this dissertation.

1.2 Document structure

The remainder of this dissertation is structured as follows~~:~~. Chapter 2 provides an overview of existing pedestrian simulation models, how they operate, and ~~what simulation frameworksare available~~available simulation frameworks. Chapter 3 ~~talks about the details of the~~ discusses the

ECS architecture ~~, its benefits, and its drawbacks~~in detail, outlining its advantages and limitations. Chapter 4 ~~goes over the details of how~~ describes the implementation of ECSMoS~~was implemented~~. Chapter 5 defines the evaluation criteria used to validate ECSMoS and ~~compares it to~~ presents a comparison with other simulators. Finally, Chapter 6 ~~provides the final remarks of the work done and concludes this dissertation~~summarizes the main findings and offers concluding remarks.

2. Current state of pedestrian modeling and simulation

Before developing the proposed simulator or comparing it with existing alternatives, it was first necessary to gain a thorough understanding of pedestrian modeling and simulation. This chapter presents a concise synthesis of the main insights obtained during the research process.

2.1 Pedestrian models

There are many aspects that include how humans move, including conscious, subconscious and physical ones.

Modeling pedestrians is a complicated ~~affair. Humans have both~~ task. There are many aspects that influence how humans move, including conscious, subconscious~~and even physical aspects that affect how one moves~~, and physical ones. Because of this, there are many competing models in use, each one with its own benefits and limitations.There are many ways of categorizing them generally. However, none is descriptive or flexible enough to capture all the aspects of the large number of models or their variations. One of the most common ways to do this is based on the scale of the entities involved [10]. Microscopic models focus on individual pedestrians and their movement, while macroscopic models avoid dealing with interactions between discrete entities, preferring to model the behaviors of whole crowds directly.

Another way of categorizing these models is proposed by [17], which divides models into the following groups:

- ~~Mechanics-based models~~**Mechanics-based models**: Inspired by continuum mechanics or force models.
- ~~Cellular automata models~~**Cellular automata models**: These interpret the world as discretized units on a grid. Agents operate under defined rules that determine how they move on the grid.
- ~~Stochastic models~~**Stochastic models**: Use random or probabilistic models to determine behavior.
- ~~Agency models~~**Agency models**: Pedestrians are treated as agents that can sense and reason about the world. They make choices about where to move depending on their perceptions of the outside world.
- ~~Data-driven models~~**Data-driven models**: Based on data collected from experiments and other sources. Uses that for building and calibrating a model.

However, both kinds of divisions are descriptive, not prescriptive. Models routinely fit into multiple or in between. Where they fall is mostly determined by the requirements and constraints of the model authors. Exotic requirements may lead to models being completely out of these constraints.

~~This dissertation will focus mostly~~ Still, such classification has its uses. For example, cellular automata models and mechanics-based models generally operate with very different modes. Cellular automata models, such as the ones seen on [18], are generally discrete, meaning that the positions of pedestrians are limited to those of cells on a grid, which has its advantages in terms of performance but may lead to the introduction of some kinds of artifacts to the movement of the agents. In contrast, mechanics-based models tend to be continuous. This makes the movements of its agents more representative of the ones seen in experimental data but comes with its own set of problems.

As a large-scale description and evaluation of the available models is outside the scope of this dissertation, it will primarily focus on the Social Forces Model (SFM), a microscopic, mechanics-based model originally proposed in 1995 [19]. ~~That is because many implementations of it have been proposed and added in~~ This model was selected because numerous implementations have been developed and integrated into various simulators over the years~~. Providing an even ground for comparisons between simulators~~, providing a more equitable basis for comparison between different simulators. Moreover, the SFM is well established and extensively studied, which facilitates the identification of potential implementation issues. ~~Its behavior is known and well understood.~~ In particular, the version described in [5] will be used, as it ~~is more similar to~~ closely resembles the implementations present in ~~today's~~ contemporary simulators.

The SFM describes interactions between pedestrians and the environment as forces. Pedestrians are modeled as circles, and, at each point in time, certain forces are applied upon them and are used to compute their acceleration, velocity, and ultimately position. Through careful calibration, these forces can reasonably model the behaviors of people moving. The original model defined three main kinds of forces. However, some implementations may add more forces, taking into consideration other aspects.

The first kind of force to take into account is the driving or motivation force, computed by Equation 1. This is a force that attempts to move the pedestrian in the direction of its ~~ultimate~~ destination. If a lone agent were placed in an empty plane and given a target destination, the motivation force would be a constant vector pointing to its destination. If there are obstacles on the way, the direction of the force might change to steer the pedestrian into the shortest available path. Which is the shortest path needs to be computed by some other means, as the model itself does not specify how.

$$\vec{F_i^{drv}} = \frac{v_i^0 \vec{e_i^0} - \vec{v_i}}{\tau} \tag{1}$$

where:

$\vec{F_i^{drv}}$ is the driving force on pedestrian i

v_i^0 is the desired speed: the speed at which the pedestrian i would prefer to walk

e_i^0 is a unit vector pointing to the desired direction

$\vec{v_i}$ is the current speed of pedestrian i

τ is the reaction time: the time it takes pedestrians to notice changes in their surroundings

The environment around an agent may also impose repulsive forces upon them. Pedestrians naturally attempt to keep some distance between themselves and others both for comfort and to guarantee the ability to continue moving forward. This is modeled as a repulsive force between them. The force between pedestrians i and j is symmetrical between them, and the final force (see Equation 2) on i is the sum of all repulsive forces applied by other pedestrians (see Equation 3).

$$\vec{F_i^{rep}} = \sum_i \vec{f_{ij}} \tag{2}$$

$$\vec{f_{ij}} = [A_i e^{\frac{r_{ij}-d_{ij}}{B_i}} + kg(r_{ij}-d_{ij})]\vec{n_{ij}} + \kappa g(r_{ij}-d_{ij})\Delta v_{ji}^t \vec{t_{ij}} \tag{3}$$

where:

$\vec{F_i^{rep}}$ is the total repulsive force from other agents on i

$\vec{f_{ij}}$ is repulsive force from j on i

$r_{ij} = r_i + r_j$ is the sum of the radius of i and j

d_{ij} is the distance between the center of i and the center of j

g is the contact distance between the pedestrians. If they are not touching, it is 0; otherwise, it is the distance between the center of i and the center of j

$\vec{n_{ij}}$ is a unit vector pointing from j to i

$\vec{t_{ij}}$ is a unit vector perpendicular to $\vec{n_{ij}}$, rotated counterclockwise

$\Delta v^t_{ji} = (\vec{v_j} - \vec{v_i}) \cdot t_{ij}$ is the tangential velocity difference of i and j

~~A, B, k, κ~~

A_i, B_i, k, κ are calibration constants

In practical terms, the repulsive force has two main components: a normal and a perpendicular vector. The normal ~~forces~~ force pushes two pedestrians away from each other ~~, increasing~~ and increases depending on how close they are. ~~If the pedestrians are close enough to touch, that force additionally~~ When pedestrians come into contact, this force increases by a secondary factor that is dependent on how much they intersect. In this way, the normal component captures both the psychological desire for personal space and the physical constraints ~~preventing pedestrians from packing too tightly. The~~ that prevent pedestrians from occupying the same area. On the other hand, the perpendicular vector represents sliding forces. When two pedestrians ~~touch~~are in contact, there is some friction between them~~that acts to slow them down. Friction acts perpendicularly~~, opposing their relative motion. This force acts perpendicular to the normal and ~~against~~ counter to the direction of movement~~. These forces can be seen~~, effectively slowing pedestrians down. These force components are illustrated in Figure 1.

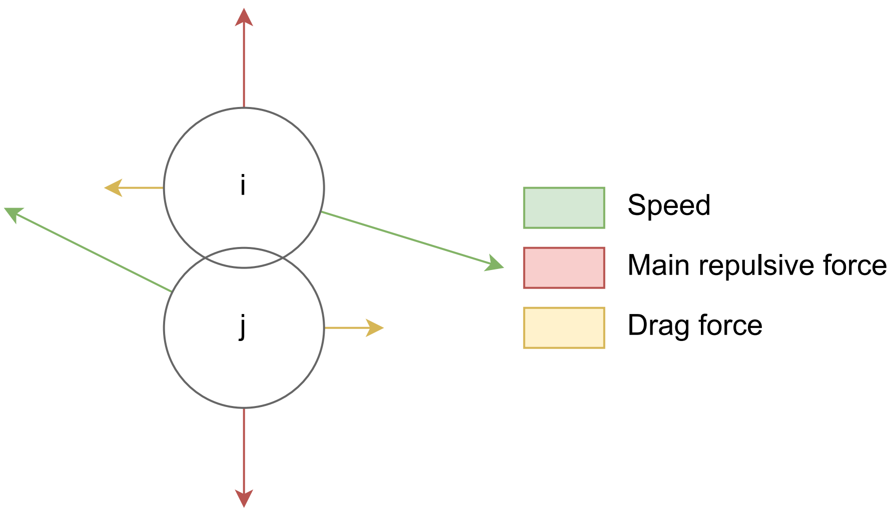


Figure 1 – Diagram of repulsive forces between pedestrians i and j

The final class of force to consider is the obstacle force. Much like the previous repulsive forces, agents also attempt to keep a certain distance ~~from themto walls~~between them, walls, and other obstacles in their surroundings. If they ~~touch~~ contact those objects, there is also an additional factor that increases the normal repulsion and a factor describing friction. However, most obstacles are not perfect circles, so the function uses the closest point of the obstacle for distance measurements. Figure 2 illustrates how this force affects agents, while Equation 4 and Equation 5 show how this force is computed.

$$\vec{F_i^{obst}} = \sum_o \vec{f_{io}} \tag{4}$$

$$\vec{f_{io}} = [A_i e^{\frac{r_i - d_{io}}{B_i}} + kg(r_i - d_{io})]\vec{n_{io}} + \kappa g(r_i - d_{io})(\vec{v_i} \cdot \vec{t_{io}})\vec{t_{io}} \tag{5}$$

where:

$\vec{F_i^{obst}}$ is the total force from obstacles on agent i

$\vec{f_{io}}$ is the force from obstacle o on i

r_i is the radius of i

d_{io} is the minimum distance from the center of i to any point of o

g is the contact distance between the pedestrian and the obstacle. If they are not touching, it is 0; otherwise, it is equal to d_{io}

$\vec{n_{io}}$ is a unit vector pointing from the center of i to the closest point of o

$\vec{t_{io}}$ is a unit vector perpendicular to $\vec{n_{io}}$ rotated counterclockwise

$\cancel{A, B, k, \kappa}$

A_i, B_i, k, κ are calibration constants

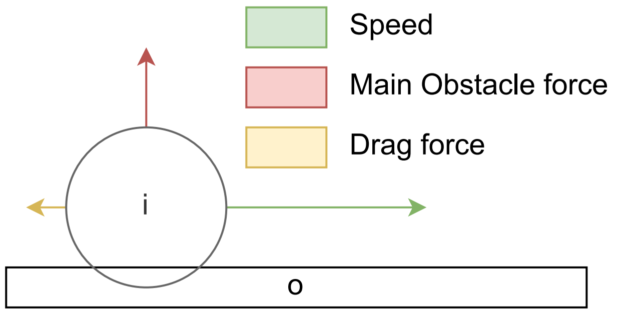


Figure 2 – Diagram of obstacle forces between pedestrian i and obstacle o

In each simulation step, all of these forces are computed for each agent~~, then~~. Subsequently they are used to compute its new ~~speed~~ velocity and position. Using basic Newtonian physics, ~~equations~~ Equations (6) and (7) can be derived. Once the agents are moved to their new locations, the simulation forces are calculated once again. This process ~~repeats~~ is repeated until the desired results are achieved.

$$\vec{v_i^{new}} = \vec{v_i} + (F_i^{drv} + \frac{\vec{F_i^{rep}} + \vec{F_i^{obst}}}{m_i}) \cdot \Delta T \tag{6}$$

$$\vec{pos_i^{new}} = \vec{pos_i} + \vec{v_i^{new}} \cdot \Delta T \tag{7}$$

where:

$\vec{v_i^{new}}$ is the new ~~speed~~ velocity of the agent i

$\vec{v_i}$ is the previous ~~speed~~ velocity of i

$\vec{F_i^{drv}}$ is the driving force on pedestrian i

$\vec{F_i^{rep}}$ is the total repulsive force from other agents on i

$\vec{F_i^{obst}}$ is the total force from obstacles on agent i

m_i is the mass of i

ΔT is the length of the simulation step

$\vec{pos}_i^{new}$ is the new position of i

$\vec{pos}_i$ is the previous position of i

Despite its relative simplicity, this model is quite robust and is capable of replicating certain phenomena that are seen in empirical studies. For example, it replicates the "Faster-Is-Slower Effect", which occurs when the flow rate of pedestrians through an obstruction paradoxically decreases once a certain speed is surpassed [5]. The SFM also reproduces lane formation, which is when pedestrians form lanes of people going in ~~the same direction~~ different directions when two streams of people converge in a limited space [20]. An example of what lane formation looks like can ~~on~~ be observed in Figure 3.

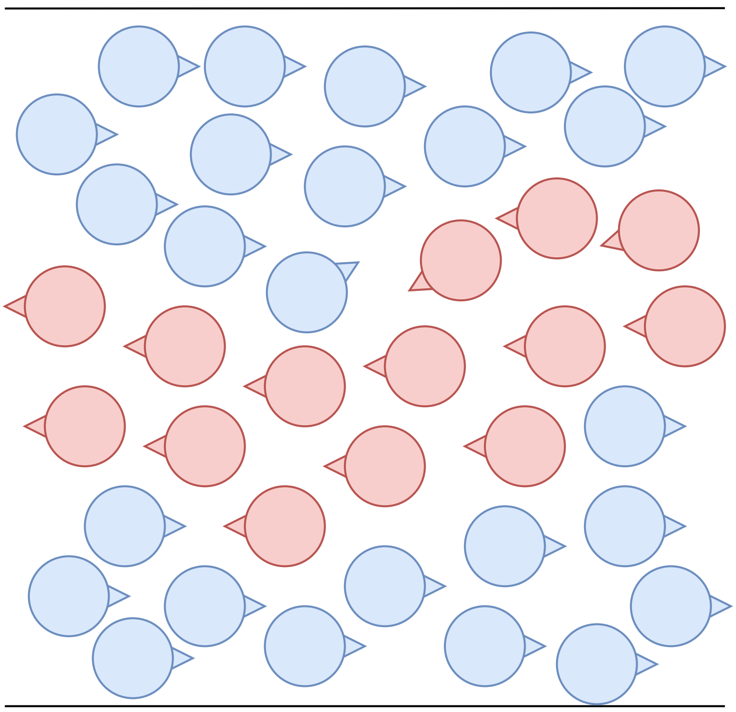


Figure 3 – Example of lane formation. Each circle represents a pedestrian ~~,~~ and the arrows represent their direction of motion. When multiple streams of pedestrians that go in different directions converge in a confined space, pedestrians going to the same destination tend to follow one another, causing the formation of lanes that go in different directions.

2.2 ~~Existing simulation frameworks~~Survey

~~Before a full establishment of~~ After examining the fundamental principles and forces that govern pedestrian behavior, it was important to understand how these models are implemented in practice. Before establishing the design requirements for ECSMoS~~was made, it was necessary to study currently available solutions~~, a study of currently available simulation frameworks was conducted. While there are many commercial products for simulating ~~the behaviorof pedestrians, they are all closed source and proprietaryfor the most part, meaning that it is~~ pedestrian behavior, most are closed-source and proprietary, making it difficult to scrutinize their results or ~~implement~~ extend them with new models. ~~Solutions like this~~ Examples of such tools include PTV Viswalk [21], MassMotion [22], Legion Simulator [23], and AnyLogic [24]. Due to these limitations, scientific research tends to focus on ~~more open simulators.~~ open and extensible simulators. As such, this work will also focus primarily on open-source frameworks that allow for detailed evaluation and implementation of new models.

To determine which simulators were most used and relevant to this work, a survey was performed. Subsection 2.2.1 defines the parameters of the survey. Subsection 2.2.2 lists the frameworks and simulators that fit into the described criteria. Finally, Subsection 2.2.3 evaluates them.

2.2.1 Survey Criteria

The following scientific article databases were searched: Google Scholar, IEEE Xplore, arXiv, SpringerLink, Science Direct and the Association for Computing Machinery Digital Library. For the search, the terms "pedestrian simulation", "pedestrian model", "pedestrian traffic", "pedestrian dynamics", "crow simulation", "crowd dynamics", "pedestrian motion", "crowd motion", "pedestrian movement", and "crowd movement" were used. The results were then filtered by date, removing anything prior to 2010. Finally, the top 50 results from each were evaluated.

After the initial search and evaluation, it was also specifically decided to add the articles from the journal Collective Dynamics to the search. This was done because, of the simulators found during the initial search, many of them were originally published there and the journal received a higher than average rate of citation on the other works. Despite this, its articles were not properly indexed by the major databases, possibly because of its small size, relative youth as a journal, and narrow focus on pedestrian and vehicular simulations.

Of the selected articles, most used custom-made software for the specific use case the researchers had in mind or did not reference a specific framework. While some of those in the former case made their code available, it was decided to not take these into account. This was done because such implementations are usually not meant to be extended by other developers. Therefore, this dissertation will only focus on simulation frameworks with the purpose of being used by other people than the authors themselves. Among these, only open-source ones with at least one update since 2015 were considered.

2.2.2 Description of available simulation frameworks

These were the simulation frameworks that passed all the filtering criteria:

Cromosim is a Python library for crowd simulations [25]. While it benefits from Python's large ecosystem of scientific tools, Cromosim has little pre-built infrastructure when it comes to helping researchers. Even basic models such as the ~~social forces model~~ SFM are not already included.

FDS+Evac is a module for Fire Dynamics Simulator (FDS) developed by the Technical Research Center of Finland for the purposes of simulating pedestrian movement in building fire scenarios [26]. It is agent-based, uses the social forces model and is mostly suited for evacuation scenarios in buildings with relatively flat floor plans. Buildings with a large degree of vertical movement within a single floor (stadiums, concert halls, and cinemas, for example) may present some challenges. However, this module has been discontinued as of version 6.7.8 of FDS.

jCrowdSimulator is a pedestrian simulator written in Java and maintained by the Fraunhofer Institute for Transport and Infrastructure Systems [27]. It can be used in the form of a library or as a stand-alone application. Currently, it only implements the social forces model and its implementation is tightly coupled to the rest of the simulator, meaning that adding new models can be a challenge.

JuPedSim is a Python package with a C++ core for simulating pedestrian dynamics [28]. Despite not having a User Interface (UI)~~[28]~~, JuPedSim is simple to use. It also includes various modules for flow/density analysis, image generation, 2D or 3D animations, reporting, and geometry generation. Model implementation is divided into three levels. (1) Tactical: route choice and short-term decisions. (2) Strategic: Long-term decisions and general objectives. (3) Operational: How to perform each action. JuPedSim can be executed inside Jupyter Notebooks for a more interactive experience. While extremely flexible for users, the fact that its codebase is split between languages and that its internal structure is tightly bound to the implemented models poses problems for developers.

Menge is a modular framework for simulating crowd movement that decomposes the problem of how to simulate human movement into four main tasks [29]. (1) Goal selection: Deciding what each pedestrian wants to achieve. (2) Plan computation: deciding on a strategy to achieve a pedestrian's objective. (3) Plan adaptation: Adapt the original plan to the current local environment of the pedestrian. (4) Spatial queries: How to retrieve data around a certain point and decide what kinds of data are useful. Figure 4 provides some additional detail on how these parts interact. To add a new model to the simulator, it is necessary to break it down into these four components while also obeying the other interface restrictions. If the new model is similar enough to another in one of these areas, it is possible to reuse that part of the implementation. However, implementing models that do not fit well into these divisions can be challenging. Menge includes a UI and various examples of its basic features. However, its last stable version was released in 2017 and all development stopped in 2019.

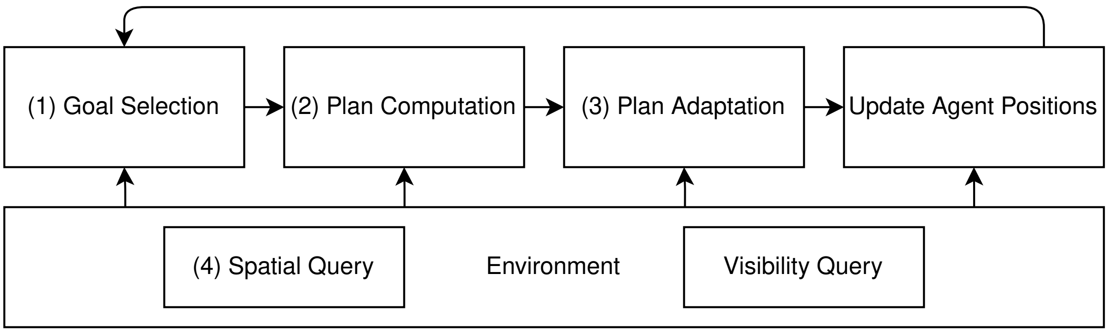


Figure 4 – Menge architecture and simulation flow. Adapted from [29].

Vadere is a simulation framework for comparing different locomotion models developed by the Munich University of Applied Sciences [10]. It was designed to be lightweight while still remaining relatively flexible. It contains both a simulation engine that can be executed via command line or via its own UI. As shown in Figure 5, Vadere follows the Model View Controller (MVC) architecture where the UI serves as the view, the state is the model and the simulator core is the controller. Inside the simulation core, models are not implemented following any specific division of responsibilities like those seen in other simulators. Instead, it uses a more integrated approach. ~~However,~~ Vadere was designed to use a mechanism called floor fields to compute the paths of pedestrians. While this implementation has its benefits [30], it imposes a performance drop on cellular automata models [10]. Attempting to use other approaches would require rewriting large portions of its internal workings.

MomenTUMv2 is a microscopic agent-based pedestrian simulator written in Java and developed by the Technical University of Munich [31]. Most of its inner workings are implemented in the form of models, which are thought of as "operation providers". These are implementations of the Strategy and Template Method software patterns. ~~This~~ MomenTUMv2's modularity is very broad, including configuration, pedestrian generators/removers, analyzers, and much more. Models that do not fit into any predefined category can be added in a special section. When compared to other solutions such as Vadere, it is much more adaptable, but it comes with a cost in terms of complexity and implementation time. One of its main benefits is that it allows the use of hybrid models, meaning models that connect two or more pre-existing models of the same layer to create behavior that is a mixture of them depending on the circumstances. Finally, it has seen very little change since 2018 and its last update was in 2020. Since then, there has been no development activity and its lead developer no longer works at the Technical University of Munich. The most recent available versions of the code are not able to be executed due to the incomplete implementation of certain features. A simplified version of MomenTUMv2's architecture and execution flow can be seen in Figure 6.

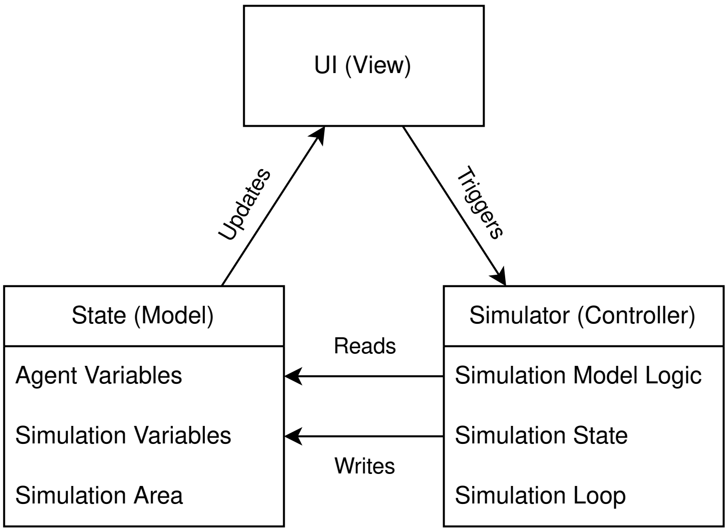


Figure 5 – Vadere architecture. Adapted from [30]

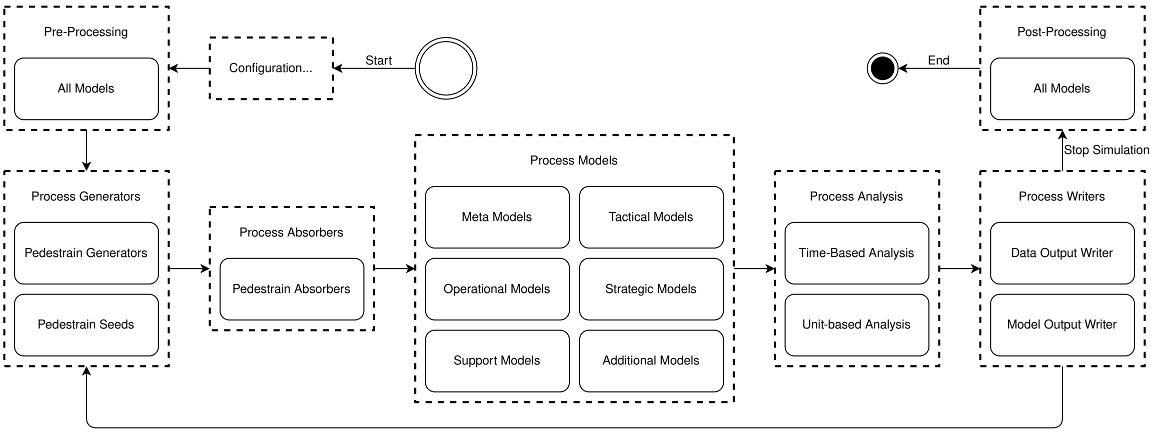


Figure 6 – Simplified MomenTUMv2 structure and flow. Adapted from [31]. Dashed rectangles represent organizational units. Solid rectangles represent a set of generic models or sets, which may contain many other families of sub-models within.

Simulation of Urban Mobility (SUMO) is a traffic simulator widely used in traffic engineering and related topics [32]. While it is mostly focused on vehicular movement, it does have some built-in mechanisms to simulate pedestrian movement. However, these were built with the purpose of enriching the traffic simulations and are very limited. For example, pedestrians are not allowed to freely roam in the 2D plane; they are restricted to certain lines/paths. For more robust simulation capabilities, it is possible to use JuPedSim together with SUMO. However, this integration is still a work in progress.

~~Finally, there are other libraries such as **Mesa** [33], **Agents.jl** [34], and many others, which focus on agent-based modeling. While these can be used for pedestrian simulations, they have little to no pre-built infrastructure for model authors, requiring a full implementation to be done for any comparison. However, due to their larger audience, they may have better tools for analysis and documentation that other purpose-built simulators. contains a summary of the considered frameworks and their characteristics. It is worth noting that projects with no clear status and no recent activity were classified as inactive. Only projects that have been officially terminated were classified as discontinued.~~

Name	Languages	Has UI	License	Last Update	Project Status
Cromosim	Python	No	GPL-3.0	2024	Supported
FDS+Evac	Fortran	Yes	Custom	2022	Discontinued
jCrowdSimu-lator	Java	No	LGPL-3.0	2024	Supported
JuPedSim	C++~~and~~/Python	No	LGPL-3.0	2025	Supported
Menge	C++	Yes	Apache-2.0	2019	Inactive
Momen-TUMv2	Java	Yes	Custom	2020	Inactive
SUMO	C++~~and~~/Python	Yes	EPL-2.0	2025	Supported
Vadere	Java	Yes	LGPL-3.0	2025	Supported

Table 1 – Summary of available simulators

2.2.3 ~~Evaluation of available simulation frameworks~~

Finally, there are other libraries such as **Mesa** [33], **Agents.jl** [34], and many others, which focus on general agent-based modeling. While these can be used for pedestrian simulations, they have little to no pre-built infrastructure for model authors, requiring a full implementation to be done for any comparison. However, due to their larger audience, they may have better tools for analysis and documentation than other purpose-built simulators. Table 1 contains a summary of the considered frameworks and their characteristics. It is worth noting that projects with no clear status and no recent activity were classified as inactive. Only projects that have been officially terminated were classified as discontinued.

2.2.3 Analysis of available simulation frameworks

Despite the variety of available tools, most of these are only compatible with a subset of models. Menge has a well-defined structure that facilitates development, but it only works well when a model can be split into its four types of tasks. Vadere and JuPedSim are very popular in general. However, their internal architecture is too monolithic, preventing code from being effectively shared between pre-existing models and new implementations. MomenTUMv2 is possibly the most modular. However, this is achieved via a lot of complexity within its architecture. Finally, libraries like Agents.jl are very capable of adapting themselves to many different kinds of models but lack any infrastructure for pedestrian simulations specifically.

As a result of these limitations, many researchers in the field end up creating their own ~~purpose built~~ purpose-built simulations. This makes comparison much harder, as certain implementation specificities can have impacts that are difficult to predict and take into account in a comparison. Removing these limitations could greatly improve comparability and convenience for model authors and one of the possible ways of doing this is by using ECS.

3. The Entity Component Systems architecture

The ECS architecture is a data-oriented design pattern or framework sometimes used in the game development industry due to its flexibility, modularity, and high performance [14]. However, it is difficult to find concrete statistics when it comes to its use, as most games are closed source and their inner workings are not widely discussed. It also has a loose definition and is sometimes confused with other similar software architectures that are also common in the area, most notably Entity-Component (EC). Additionally, the various implementations can be substantially different due to their underlying technologies and objectives. Finally, there are few academic sources for this topic. Most works on this subject are blogs, videos, and personal anecdotes. Because of these factors, this section will discuss the most common aspects of the ECS architecture, saving details of the specific implementation used for ECSMoS discussed in Chapter 4. Nevertheless, this architecture has the potential of addressing many of the problems raised in Section 2.2 and merits further exploration.

3.1 General structure

Generally speaking, in an ECS implementation, most ~~things~~ elements are divided into the following three main kinds of concepts:

Entities represent general objects in the world. For example, in a crowd simulation, each pedestrian would be an entity, but so would obstacles and other things that might affect them. Usually, they consist of a simple identifier that can be used to group other kinds of data. Entities usually do not carry additional data, but many implementations allow entities to be "disabled", meaning that they are not used/visible to other parts of the architecture.

Components can be considered the attributes of entities. Each component is associated with only one entity and stores the data relevant for a certain aspect of its behavior. In most implementations, each entity may also only have one component of each type associated with it. Following the previous example, each pedestrian would have a position component and a ~~speed~~ velocity component, each of which contains all the data necessary to characterize the entity on the relevant aspect. Entities may have as many components as necessary to describe their behavior. However, components do not contain logic. On the other hand, some components also do not store data within them; their presence is already the data itself. In this situation, the component works like a tag, marking the entity as possessing a certain property. For example, an entity that is a pedestrian may have an empty `Pedestrian` component, which serves only as a tag for filtering.

Systems are processes that can read and modify components. They are the main place for logic in this architecture. They usually have a well-defined singular purpose and can access all data as though it were global. For example, a simple system might move entities to their next position according to their ~~speed~~ velocity component. Systems are executed in a predefined order that may be specifically determined by the developer or derived by some set of rules. One common implementation of rules-based ordering is to define the dependencies of each system, meaning systems that must be executed before each one starts. This allows the framework to define by itself when each system is executed. Once all systems have been executed, the cycle is started once more from the first system, creating a loop. Systems are also usually capable of filtering the entities that they want to read data from or operate on by using the presence or absence of certain components.

Figure 7 contains an example of how these three concepts interact. Another way of understanding these concepts is to compare them to a relational database. In this paradigm, each component type has its own table and uses the entity it is associated with as a primary key. A system then would perform queries for the data in the database. The entity information can be used for joining data from

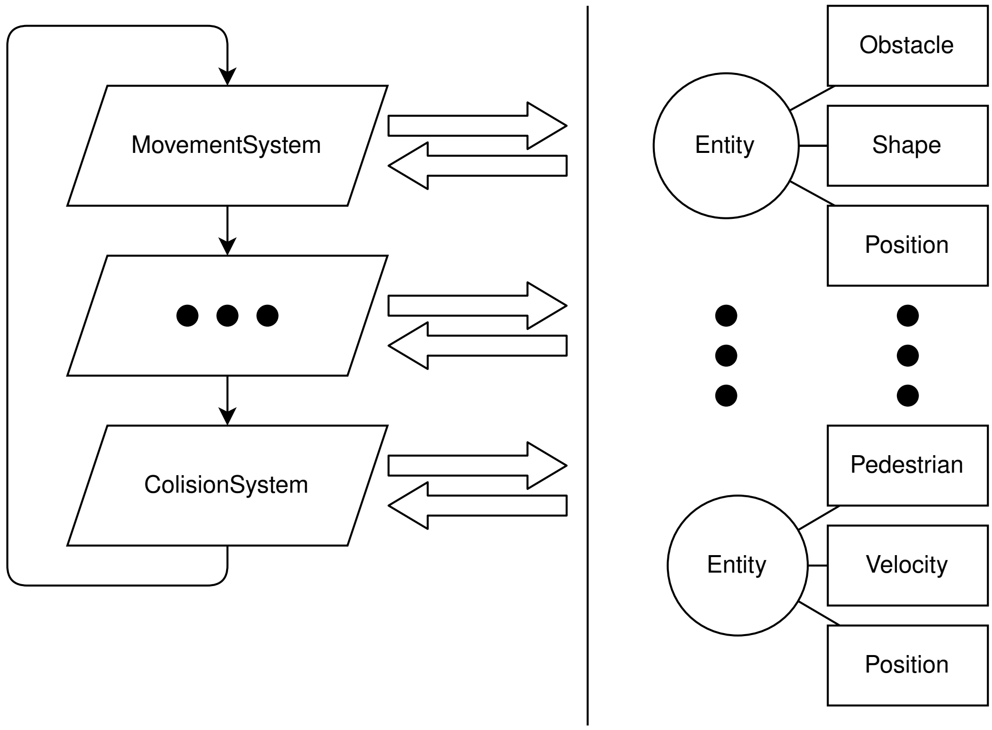


Figure 7 – Example of how an ECS project is divided. On the right, there are entities and their related components. On the left, systems are shown. Systems are executed according to their specified order and read/write data to and from the components.

different component tables and the presence or absence of each component can be used to filter out certain entities. In this way, systems can perform operations only on entities that are relevant to the systems function. A system may also write back to the database and following systems can access the newly written data. This general structure can be seen in Figure 8.

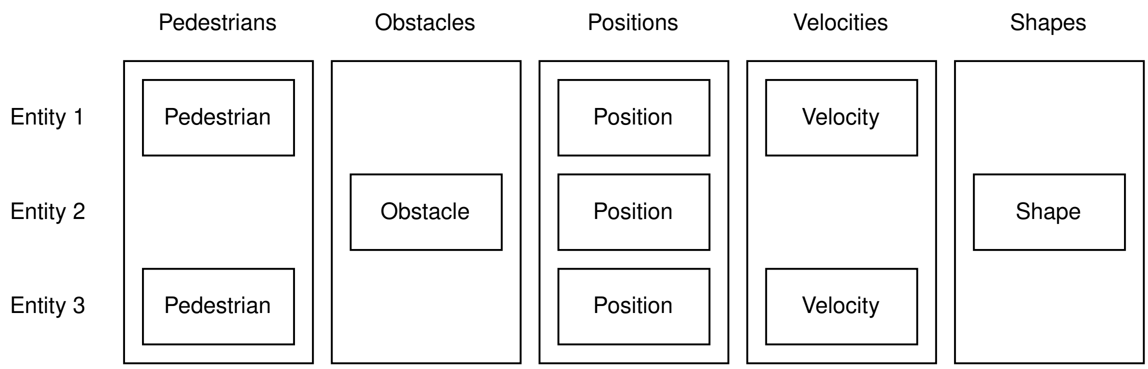


Figure 8 – Example of how an ECS can be seen as a database. Adapted from [35].

3.2 Component storage strategies

While all ECS implementations have the same basic structure, one of the main differentiating factors between them is the underlying method for storing the component data and their connections to an entity. Since querying entity and component data is one of the most performed operations on this paradigm, having a well-thought-out method for storing and retrieving such data is of utmost importance. The following is a non-exhaustive categorization of ECS implementations when it comes

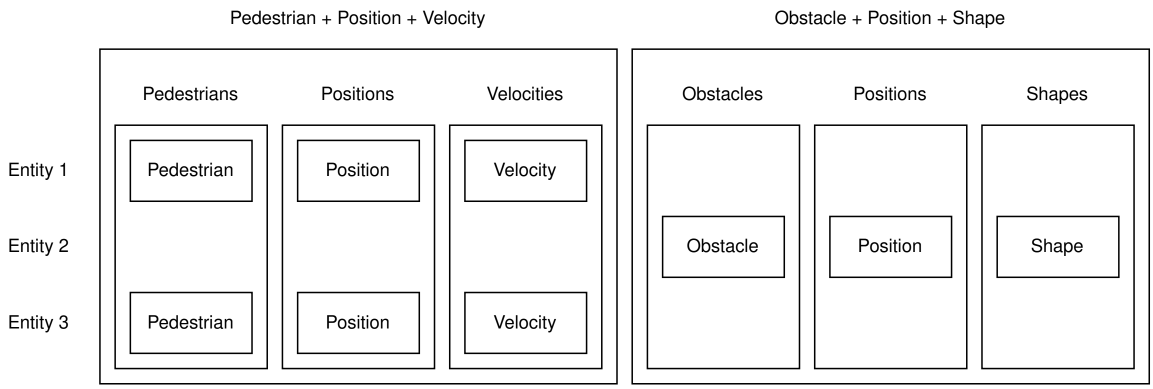


Figure 9 – Example of how archetype-based ECS stores components.

to this aspect:

Archetypes: Implementations that rely on archetypes group entities based on the components they have. Each combination is called an archetype and is stored in a contiguous area of memory, often as a struct of arrays, meaning that they have great cache spatial locality. When a system requires data, the framework checks which archetypes conform to the requirements and provides the data. This strategy has the best iteration performance of the discussed implementations. However, adding or removing components from an entity forces it to move all the data related to that entity to another archetype. This reduces the efficiency of such operations. Examples that mainly use this approach include Flecs [36], Unity DOTS [37], Bevy ECS [38], Legion [39] and Hecs [40]. For a visual example, see Figure 9.

Sparse set: In sparse set implementations, a sparse set is maintained for each component type. Sets use entities as keys to retrieve components and the framework checks all the sparse sets for the components each system requests to gather the data. As a result, iteration over the entities is much slower due to lack of cache locality. However, adding or removing components becomes much faster. Entt [41] and Shipyard [42] are some of the most well-known frameworks that use sparse sets. This approach is the most similar to the one depicted in Figure 8.

Bitset-based: As the name suggests, bitset-based implementations work by using a bitset to ~~say if~~ communicate whether or not an entity has a certain component. This can be done by having arrays that contain the component data and a bitset for each entity, or it can use other data structures such as hierarchical bitsets, which may provide better iterative performance. EntityX [43] and Specs [44] are some notable examples in this area.

Due to the many benefits and drawbacks of each approach. Some ECS frameworks allow for the storage type to be defined on a component-by-component basis. Such an approach allows developers to maximize the performance of the system. Generally speaking, most components should be stored in something similar to what the archetype implementation uses, which gives the best possible iterative performance. However, components that are frequently added and removed in short succession can be stored similarly to sparse set or bitset implementations to reduce the overhead of these operations. Examples of frameworks that allow for such a nuanced approach include Bevy ECS [38] and Specs [44].

3.3 Performance characteristics

Depending on the details of how components are stored, this architecture maximizes the efficiency of the CPU cache when compared to the architectures of other simulators, which gives it a substantial

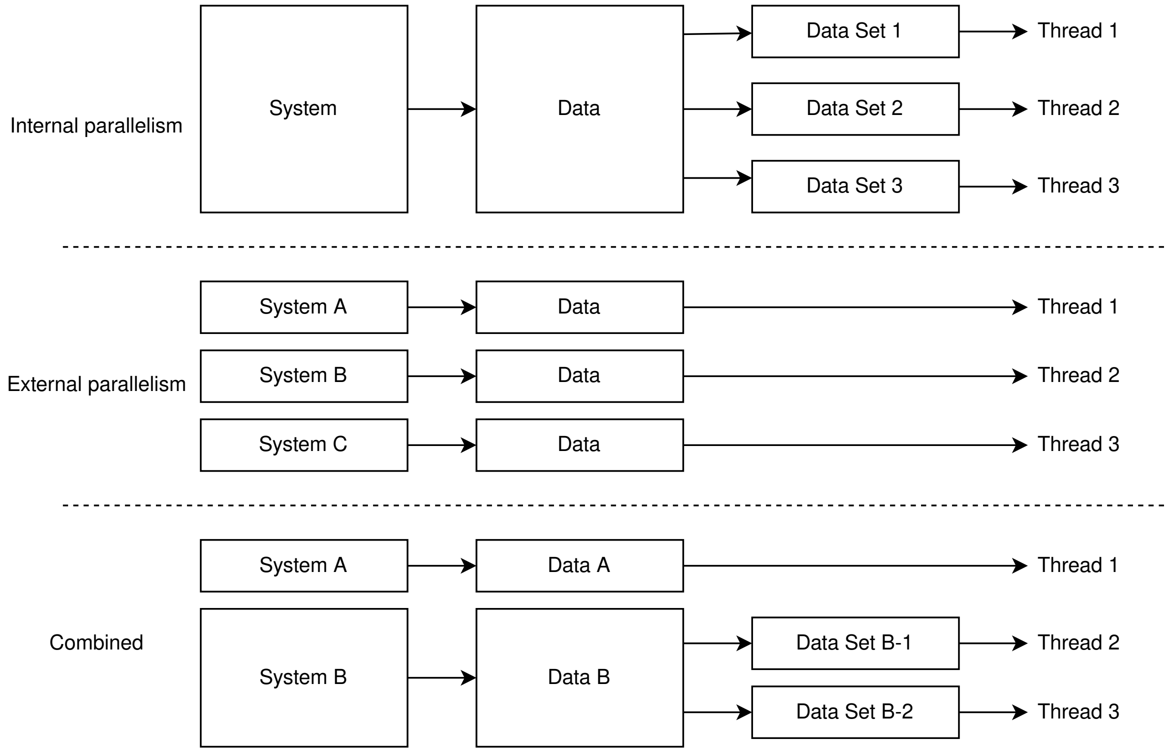


Figure 10 – Example of how internal and external parallelism work and how they operate when combined.

performance advantage. This happens because the data for each component type can be stored in a single place, usually an array, and can be directly loaded into the cache when a system starts its execution, meaning that it has high cache spatial locality. As a result, the CPU does not need to check the RAM as often, which reduces the time it takes to retrieve data and consequently improves performance. For comparison, in simulators such as Vadere, the update loop first goes through each entity and then it is able to load the data for each individual property. As a result, the CPU has to go through noncontinuous regions of memory, increasing the number of cache misses.

In addition, the fact that systems are independent allows for a certain level of parallelism to take advantage of multithreaded systems. This comes in two forms: internal and external parallelism. External parallelism is when two or more systems may be executed at the same time. This can happen when the systems do not modify any pieces of data that are used by each other and there is no logical requirement for their execution to be sequential. Such an approach is useful when multiple kinds of data are necessary for one operation, but computing them is time-consuming. The expensive computations can be split off into multiple different systems that save their data to intermediate components. After all these computation systems have concluded their execution, the final system gathers the intermediate components and performs the operation.

Internal parallelism can occur when it is possible to process each entry independently of each other. In this case, a system can split the data it has received into multiple groups that can be processed in parallel. These two modes of parallelism can also work together. In this situation, systems can be executed in parallel and systems can split their data into different parallel processes. Figure 10 shows an example of these approaches.

While equivalent techniques could be employed on existing simulators, such changes would be challenging to archive and prone to causing issues. In ECS, these changes are much simpler. Internal parallelism is possible in a large portion of cases and it is much easier to implement, although it may not be worth it for simple and inexpensive systems. External parallelism is somewhat harder because it requires an analysis of the inputs of the systems to prevent concurrent modifications to the same piece of data or to handle it gracefully. However, some ECS frameworks, such as Bevy ECS [38], allow for a seamless implementation of these concepts.

3.4 Benefits and drawbacks

In addition to the performance benefits, ECS promotes adaptable and reusable design. Components work best when they represent a singular aspect of each entity and are elementary. Such components can be composed into multiple kinds of entities and, because the systems use the components associated with an entity to decide if they will perform operations on them, the behavior of the entities can be changed mid-simulation. For example, a pedestrian may have a `Position` and `Speed` `Velocity` component, which causes the `SpeedSystem` `VelocitySystem` to make it move according to its ~~speed~~velocity. If the pedestrian then decides to sit, the `Speed` `Velocity` component can be removed, which automatically removes it from the `SpeedSystemVelocitySystem`'s list of entities to operate on. Also, if it is desired to add other kinds of moving objects to the simulation, cars for example, it is not necessary to create new logic to handle their movement. Any entity with `Position` and `Speed` `Velocity` will automatically be affected by the `SpeedSystemVelocitySystem`.

Despite these advantages, ECS also poses some problems that have limited its acceptance even in the fields where it sees some use. The main one is the lack of familiarity of programmers who use Object-Oriented programming (OOP) and other common programming paradigms with the design principles it is built upon. For example, while ECS can be implemented in OOP-focused languages, it breaks many of OOP's principles. Making effective use of this architecture requires reframing problems to avoid some aspects, such as inheritance and polymorphism. This problem compounds on itself, as the lack of popularity means that, generally speaking, tooling for this architecture is less developed than others and there is less information about it, which naturally perpetuates its status.

ECS also has some problems inherent to its structure. Due to the fact that component data is effectively global, tracking down issues caused by improper manipulation of data is hard, as they could come from anywhere. Additionally, observing and understanding the global data is ~~a complicated affair~~ complicated during debugging, as the data for each entity is spread around multiple places and possibly using multiple forms of storage, making it difficult to have a holistic picture of ~~what~~ the system's state~~is~~.

4. ECSMoS implementation

Considering the ~~problem~~ problems of other simulators referenced in Section 2.2, the main objective of ECSMoS is to be more flexible than existing simulators. It should accommodate the addition of new models easily, should remove as many constraints for model authors as possible, allow for the creation of variants of existing models and the creation of hybrid models. Additionally, it should be at least as performant as existing solutions. While there are other factors that greatly influence the usability of a simulator, such as UI, documentation, and integrations with other tools, these were not focused upon for this work. Any implementation for these or other areas was merely made for the purpose of supporting the central goals of the project.

The main technological choice for the implementation of ECSMoS was the specific ECS framework to be used. While there was the option of creating a bespoke implementation, it would likely be much less mature, stable, and feature-rich than the ones already available. Among the available options, Bevy ECS, an ECS framework implemented in the Rust programming language, was chosen. This was due to how Rust and Bevy ECS complement each other, allowing for seamless external parallelism enabled by the compile-time guarantees provided by Rust's borrow-checker and explicit declaration of mutable values. These features allow Bevy to know at compile time which components and entities are accessed by any given system and whether they can perform changes to them. As such, a dependency graph can be constructed and Bevy can use that to prevent the execution of systems that conflict with each other (such as those that write to the same components) from being executed in parallel while allowing the non-conflicting ones to do so.

In addition to the standard ECS constituents of entities, components, and systems, Bevy also provides a few more features. **Schedules** serve as management units for systems. They defined what systems are executed, what their execution order is, and under what conditions they are active. **System Sets** are system organizational units. They can be used to group systems that have similar functions or by any other differentiating factor. **Resources** are a way of storing data globally without association with a specific entity and are useful for representing global state. Bevy also has a built-in **Events** framework, which allows communication between systems without needing to update components. Finally, Bevy has an advanced change detection method and developers are able to use that to only execute certain systems when certain components are added or changed, reducing computational load. These and many other features are all fully compatible with the Bevy's parallelization mechanisms, allowing full use of multithreaded hardware.

4.1 ECSMoS ~~Structure~~structure

To improve organization and effectively provide a division of concerns, ECSMoS is built out of modules, which are built on top of what Bevy calls plugins. Each module can add its own systems, resources, components, etc., to the simulation. Modules are not executed in any particular order. Each of their systems can individually be configured to be executed before or after others. In cases where modules have a dependency on one another, meaning that a system in module A must be executed after a system in module B, system sets can be used to define this dependency behavior.

Bellow is a list of the currently available modules and their general function:

- Default module: Enables basic internal Bevy modules useful for the general simulation framework
- Auto End Simulation module: Adds functionality for automatically stopping a simulation under certain conditions, such as when there are no more pedestrians or when a certain amount of time has passed

- Display module: Enables the UI of the simulator
- Flow Field Pathfinding module: Used to compute the best paths for pedestrians at any given point to their destination. The data provided by this module is later used by the Social Forces module to compute the direction of the motivation force. This module is heavily inspired by the floor fields used on Vadere for the same purpose.
- Kinematics module: Handles basic movement according to the laws of physics. While other modules may handle the computation of forces, this module actually moves pedestrians according to their final speed.
- Movement Tracking module: Used for collecting and exporting data related to the movement of pedestrians in the simulation.
- Simple Objective module: Used for determining if a pedestrian has arrived at its destination (called an objective inside the simulation) and what actions should be taken next, most commonly removing it from the simulation.
- Simulation Area Module: Used for defining the region where the simulation takes place. Pedestrians that attempt to leave the simulation area are prevented from leaving. The information provided by this system is also useful for flow field pathfinding, which uses it to limit the paths it needs to compute.
- Social Forces Module: Implements the social forces model for simulating pedestrians.
- Spawner Module: Used for creating pedestrians at ~~a defined region~~ defined regions at a certain frequency, providing a constant flow of agents in the simulation.
- Start Time Module: Simply records the time at which the simulation started. This information is used during the export for naming and differentiating various simulation runs.

Figure 11 shows an example of how inter-module dependencies and system sets operate. It is important to note that it does not show all systems and the general flow has been simplified for better understanding. The image shows two modules, their systems, system sets, and how they interact. The rounded boxes represent predefined system sets and the arrows between the systems and system sets represent their dependencies. Systems without a dependency, such as `Compute Obstacle Collision Map` can start execution as soon as possible and, in this example, can also be executed in parallel. Bevy executes systems in parallel in no particular order and it can be changed from iteration to iteration. The height of each system denotes the period of time when it can be executed. As shown, `Compute Motivation Force` depends on the Flow Field Pathfinding module's system set, meaning that it will only be executed after it has finished, unlike the other social forces systems.

In the event that it becomes necessary to create a new module that has systems that must be executed after the first three systems of the Flow Field Pathfinding module, but that does not need to wait for the remaining systems, the current system sets are not sufficient. In this situation, the new module could create a new system set that exists on top of the current structure. This set can then only include the desired systems and be used for defining dependencies. Figure 12 shows an example of this. This new set can be anonymous and, as such, remain in use only inside the module or be given an identifier that would allow it to be used by other modules. Either way, it has no impact on the pre-existing system set or any of its relationships.

While generally a module is added as a whole. It is also possible to add only certain parts of it. For modules where there is a clear expectation that certain parts are only needed sometimes, it is possible to pass configuration variables to the module that enable or disable certain sections. If there was no expectation of this need during development, it is possible to bypass standard module initialization and add the required parts manually. While more time-consuming, this allows a large

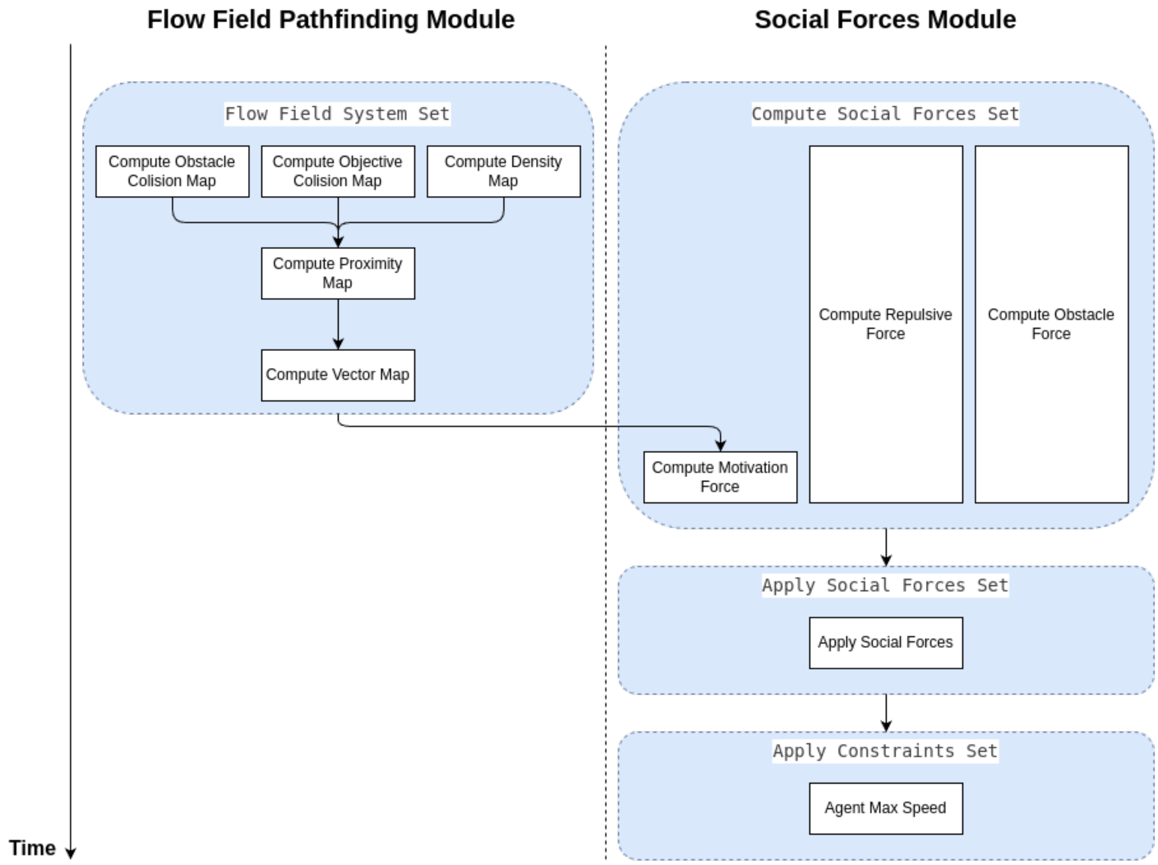


Figure 11 – Example of how ECSMoS handles inter-module dependencies

degree of control even over modules created by third parties. This is particularly useful when creating derivatives of pre-existing modules, as it allows the child module to only keep the parts of the parent that it desires while removing the others.

As a result of this approach, ECSMoS does not have a linear execution/relationship path between its modules like other simulators. Instead, it can be best imagined as a network, where each system is a node with its own relationships. Systems are generally more connected to other systems within the same module, but they often connect to other ones. While this increases the general flexibility of the simulator, it also comes with a drawback, as it allows for developers to unknowingly create circular dependency paths.

In its current state, ECSMoS has a simple UI that is useful for debugging and visualizing the results during execution. The interface is built on top of Bevy's default rendering systems, which are relatively restrictive, but it is possible to add other UI systems such as Bevy EGUI [45]. This rendering functionality is not directly tied to the simulation state. There are systems that are executed at the end of each simulation loop that translate simulation data into values comprehensible by the UI systems. As such, it is possible to add interface elements that are model-specific.

Like Vadere, ECSMoS does not have a unified data collection or output mechanism. Both use data collectors, which only store certain fields, and post processors that output the collected data to one or multiple files. In both cases, a trajectory data collector is provided that is able to track the positions of pedestrians and export them in the Comma-separated values (CSV) format or equivalent. However, more complex data collection may require additional systems to be created.

Finally, ECSMoS does not currently have a proper external file format for storing scenario definitions. Therefore, defining a scenario is currently done directly via code. Such a system is possible,

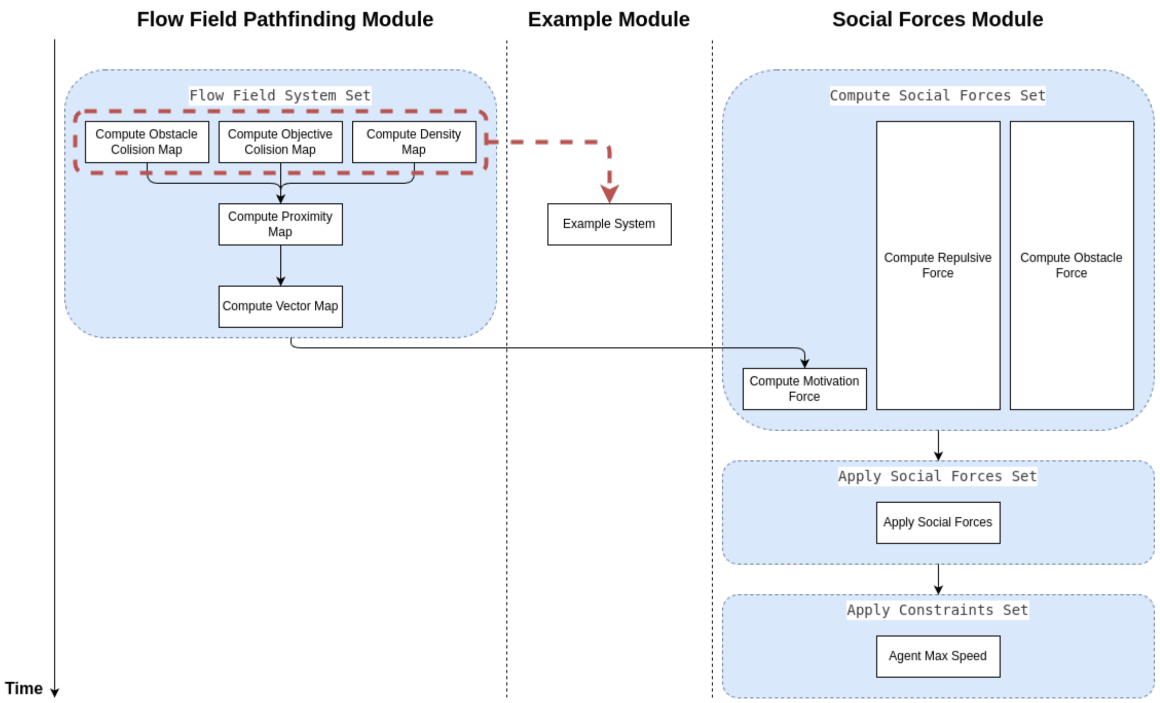
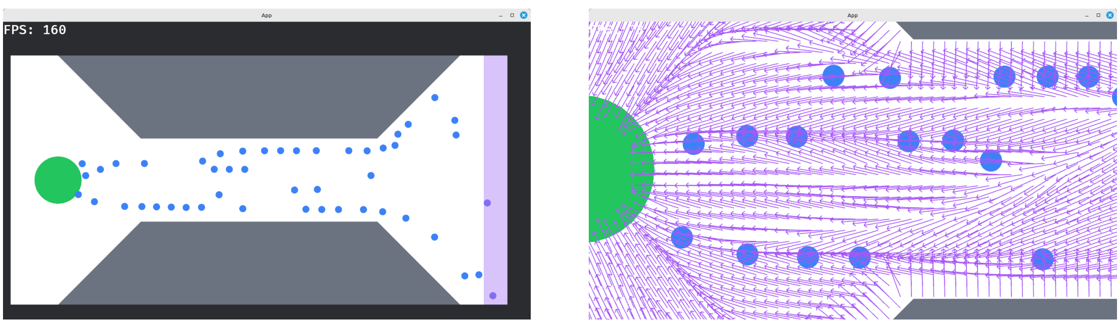


Figure 12 – Example of how ECSMoS handles inter-module dependencies where there is no predefined system set grouping only the required systems. The thick dashed and rounded square represents the unnamed set created.

but ECS creates some additional complexities when it comes to this, most notably in the definition of system dependencies. ~~While this is possible to do, it was not considered inside~~ There are many ways in which this could be accomplished, but it was considered outside the scope of the current work ~~.~~

Figure 13 contains examples of the current UI of the simulator. Figure 13a shows the general appearance of the simulator. The dots are agents, the circle to the left is their destination, and the rectangle to the right is where they are added to the simulation. On the top left, the counter shows how fast the simulation is going. Each frame is the equivalent of a simulation step. Figure Figure 13b shows one of the many debug visualization modes. The one on display shows a field of vectors that point to the direction in which a pedestrian would go if it were in that location. Other debug modes include visualization of proximity to the target destination, collision zones, and statistical density of pedestrians. These modes are very useful for understanding the inner workings of the implemented model, but they impose a significant performance penalty.



(a) General ECSMoS UI **(b)** Vector field UI

Figure 13 – ECSMoS interface

5. Evaluation

For the purposes of this work, two kinds of evaluations were defined for the simulator: qualitative and performance. The qualitative evaluation concerns itself with evaluating the usability, structure, reusability~~and other such~~, and other similar aspects. In contrast, the performance evaluation consists of measuring its performance when compared to other simulators in similar test scenarios~~,~~. Of these two, the qualitative evaluation is arguably the most important. It focuses on comparing the capabilities of the simulators when it comes to the implementation of new models and adaptation of already existing ones. As such, it ~~touches on~~ directly addresses the main objective of this project: flexibility.

In addition to ECSMoS, two other simulators were selected for the comparison: Vadere and JuPedSim. These were chosen because they are the most robust ones still being updated. Out of the remaining simulators, MomenTUMv2 would also have been a suitable candidate due to ~~how flexible its architectureis~~the flexibility of its architecture. However, the fact that its continued development is uncertain and that it was left in a non-functioning state with no clear indication of which previous version could be considered usable meant that it was decided to remove it from the comparison. Conversely, jCrowdSimulator receives regular updates but was not considered for this comparison for lacking much of the basic infrastructure of a general simulator and due to its tightly coupled nature.

5.1 Qualitative ~~Evaluation Strategy~~evaluation strategy

For qualitative evaluation, a set of ~~common scenarios seen during the development of new models~~ applications seen in the research was formalized and proposed as test ~~scenarios~~use cases. Table 2 contains a list of the ~~scenarios~~use cases accompanied by their descriptions and main objectives. These were designed to test specific aspects of each simulator. However, they also had to be somewhat flexible to avoid hitting specific weak points of the simulators that could be easily avoided by slight shifts in the definition criteria.

~~Scenario~~ Firstly, Use Case 1 contains an example of this flexibility. The simulators being evaluated do not all use the same basic pathfinding system. Since ~~what pathfinding systems are available~~ the choice of pathfinding system is not an object of study in this dissertation, it was decided to ~~write the scenario~~ define the use case in such a way as to allow for the use of any one of these. Similar accommodations have been made in the other ~~scenarios~~use cases.

~~Scenarios~~ Use Cases 1 to 3 test basic situations that are likely to ~~come up~~ arise during the development of most novel models. If ~~they are~~ a model is entirely novel, ~~they would likely go with a path more similar to scenario~~ it would likely follow a path similar to Use Case 1. ~~Models that are mostly based on another but have additional variables /factors would match scenario~~ On the other hand, models that are largely based on an existing model but incorporate additional variables or factors would correspond to Use Case 2. ~~Lastly, Scenario~~ Finally, Use Case 3 covers ~~additional~~ implementations that may not ~~cleanly~~ neatly fit into the previous categories.

~~Scenarios~~ Use Cases 4 and 6 ~~take aim at~~ target features that are less commonly used~~but that may be hard barriers when it comes to~~, but may present significant barriers during implementation. Some models explicitly consider how an agent makes decisions in a changing environment [46], while others include actions ~~that involve things other than~~ beyond movement, such as waiting in line ~~and~~ or loitering [47, 48].

~~A~~ Moreover, a simulator's pathfinding algorithm may also have limitations or performance issues. These algorithms are used to compute ~~what path the~~ the path pedestrians would attempt to take if the other agents around them were static and, for the Social Forces Model, ~~are used as a~~ serve as the basis for the driving force. Vadere and ECSMoS use similar implementations that work well for the

SFM but ~~that generally have worse performance~~ generally perform worse for cellular automata models [10]. As such, ~~it~~ modifications may be necessary ~~to change them under some set of circumstances. Scenario 5 tests that change.~~ under certain circumstances, which is tested in Use Case 5.

After the ~~scenario definitions~~ use case definitions were stated, a study was ~~made in implementing~~ conducted to implement them on each simulator. Implementations were then graded according to three metrics: implementability (~~meaning~~ ease of implementation), reusability, and complexity. These grades were given on a three-level scale~~, going from poorto good. The details~~ : poor, fair, and good. Details of each level are described in Table 3. In summary, a grade of "good" indicates that a simulator can easily handle the implementation of the ~~scenario. Its architecture poses minimal challenges in regard to that criteria. Conversely, a~~ use case, with minimal architectural challenges. A grade of "poor" implies significant challenges, ~~to the point that it may not be feasible to implementthe scenario~~potentially making the use case infeasible to implement. Finally, a grade of "fair" indicates ~~that there are some challenges , but these are possible to circumvent~~ manageable challenges that can be overcome with a reasonable level of effort.

~~№Name Description Objective 1 New Model Add a new simple model that uses a basic pathfinding system already implemented on the simulatorTest reusability of basic simulator infrastructure2 Model variation Create a variation of the Social Forces Model available in the simulator with an additional attractive force that guides agents to areas where pedestrians would naturally concentrate, as defined in [19]Test reusability of existing model implementations when the variation is small3 Hybrid model Create a model based on an already existing one that changes how it operates under some set of circumstances, similar to models that take panic into account, like those in [13] Test general adaptability of already existing models to larger changes4 Procedural environment changes Create a simulation environment that changes over time, adding new obstacles that change the pedestrian's paths as seen on [46] Determine whether the simulator is able to handle updates to data that is not related to the pedestrians directly5 Implementing a new pathfinding algorithm Add a new algorithm for determining the shortest route between a pedestrian's current location and their destinationCheck the flexibility of current models to changes in base infrastructure, which may be necessary for performance or other limitations [10]6 Queuing and loiteringAdd procedures for pedestrians forming queues and for pedestrians staying still while waiting for something, but with some level of reactivity to the surrounding environmentQueuing and loitering are behaviors that significantly change how a pedestrian moves and interacts with its environment. They also break many assumptions that are usually made by models, which often require continuous movement [47]List and description of test scenarios.~~

5.2 Performance ~~Evaluation Strategy~~evaluation strategy

When it comes to scientific pedestrian simulation software, performance is often a secondary concern. For many ~~such~~ applications, the total ~~amount of time it takes for the simulator to run is of little concern~~ runtime of the simulator is acceptable as long as it does not impede or unreasonably slow down the research process. ~~Considering these factors~~Given this context, the main objective of this evaluation is to measure the general effect of the change in architecture ~~to~~ on the performance of the simulated models, ~~not to perform a comprehensive~~ rather than performing a comprehensive benchmark comparison.

~~As there are~~ Given no standard scenarios ~~used~~ exist for performance comparison between pedestrian simulators, it was necessary to develop new ones. ~~Following the described objectiveof this evaluation~~In

№	Name	Description	Objective
1	New Model	Add a new simple model that uses a basic pathfinding system already implemented on the simulator	Test reusability of basic simulator infrastructure
2	Model variation	Create a variation of the SFM available in the simulator with an additional attractive force that guides agents to areas where pedestrians would naturally concentrate, as defined in [19]	Test reusability of existing model implementations when the variation is small
3	Hybrid model	Create a model based on an already existing one that changes how it operates under some set of circumstances, similar to models that take panic into account, like those in [13]	Test general adaptability of already existing models to larger changes
4	Procedural environment changes	Create a simulation environment that changes over time, adding new obstacles that change the pedestrian's paths as seen on [46]	Determine whether the simulator is able to handle updates to data that is not related to the pedestrians directly
5	New pathfinding algorithm	Add a new algorithm for determining the shortest route between a pedestrian's current location and their destination	Check the flexibility of current models to changes in base infrastructure, which may be necessary for performance or other limitations [10]
6	Queuing and loitering	Add procedures for pedestrians forming queues and for pedestrians staying still while waiting for something, but with some level of reactivity to the surrounding environment	Queuing and loitering are behaviors that significantly change how a pedestrian moves and interacts with its environment. They also break many assumptions that are usually made by models, which often require continuous movement [47]

Table 2 – List and description of use cases

Criteria	Grade	Description
Implementability	Good	The simulator can easily accommodate the required additions. It has a clear place and method for receiving the new model, requiring minimal changes to existing code.
Implementability	Fair	Implementation is feasible but requires moderate exploration of the code to identify where changes are needed or there are some tight coupling/rigid designs that slow down development.
Implementability	Poor	Simulator lacks extension points for this ~~scenario~~use case, requires deep internal changes or requires modification to parts of its code that would be very difficult to decouple. Implementation is slow, messy or may not be feasible.
Reusability	Good	Existing code can easily be reused in this ~~scenario~~use case. The related models, utility methods, etc. are generic, have good abstractions, and are composed of easily swappable units.
Reusability	Fair	Some reuse is possible, but components are partially coupled to specific behaviors or ~~scenarios~~use case. Reusing core infrastructure may impose minor restrictions or require small modifications. Model-specific logic may need to duplicate existing features if interfaces are too narrow or rigid.
Reusability	Poor	Simulator components are hard-coded or specialized for specific agents or ~~scenarios~~use cases. Implementing the scenario requires rewriting or duplicating most, if not all, of a model or internal system to operate.
Complexity	Good	Implementing the ~~scenario~~ use case is possible with minimal cognitive effort. The changes are well isolated, have logical continuity, touch as few other systems and the data/execution path they follow is direct and predictable.
Complexity	Fair	Implementing the ~~scenario~~ use case requires working with several components whose relationships aren't always intuitive. Changes may need to be performed in many regions of the code. Understanding the control flow is unintuitive or time-consuming, but possible.
Complexity	Poor	Fully integrating the ~~scenario~~ use case is very complicated. Control flow is chaotic, multiple tightly coupled parts have to be modified, or they may require a large redesign of the architecture of the simulator itself.

Table 3 – Description of evaluation criteria and their levels~~.~~

line with the stated objective, minimizing the effects of other factors was paramount. Firstly, ~~complicated~~ complex topologies disadvantage simulators ~~like~~ such as JuPedSim, which determine base routes via ~~a method that requires~~ polygon triangulation. The more polygons there are in a ~~situation~~topology, the slower it is to compute the base paths of pedestrians. ~~On the other hand, largespaces, even if empty~~

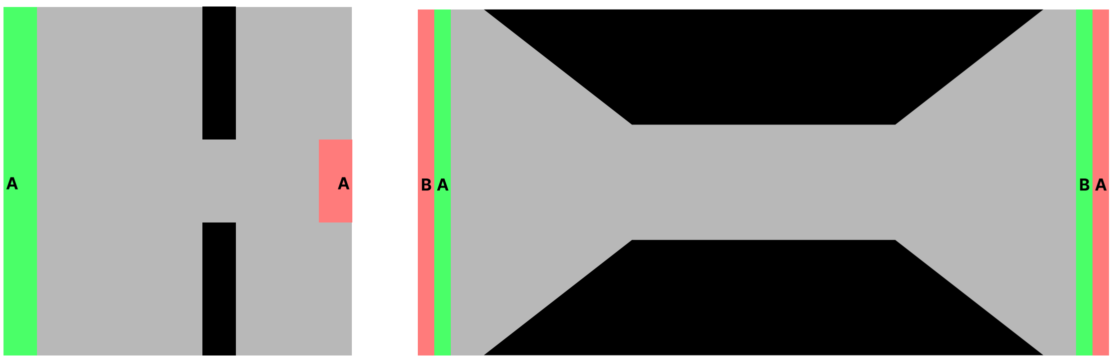
Conversely, large, empty spaces substantially increase the number of computations ~~that~~ for simulators like Vadere and ~~ECSMoS require~~ECSMoS. These factors~~are independent of architecture and~~ , independent of the underlying architecture, can cause non-linear performance drops that ~~can change both depending on the topology and the number of pedestrians in the simulation. As a result, it was decided to create scenarios that~~ vary with both topology and pedestrian count. Therefore, scenarios were designed to minimize these effects.

~~This evaluation focuses on how each simulator handles an increasing number of pedestrians in a few scenarios. The simulations start with zero agents on the field and increasing amounts are added. Once each agent arrives at its objective, it is removed from the simulation. Other than topology, the main factor that varies from simulation scenarios is the frequency with which pedestrians are added, designated f_p, measured in $pedestrians/s$ and varying from 0.5 to 8. The low values of f_p mimic areas with low but active pedestrian traffic, the intermediary ones slowly go towards saturation and, at high values, the equivalent of traffic jams for pedestrians starts to happen.~~

~~The Social Forces Model was used for all experiments and the simulator constants were standardized to those seen on or equivalent values in each simulator. These include the calibration constants see on , and . As there is no clear equivalence between the underlying algorithms for computing the shortest path between a pedestrian and its destination, a piece of data crucial for the computation of the motivation force, these have been left at their default values.~~

~~Parameter Value UnitA_i 2000 $N B_i$ 0.08 $m k$ 120000 $\frac{kg}{s^2} \kappa$ 240000 $\frac{kg}{m \cdot s} r_i$ 0.3 $m \tau$ 0.5 $s v_i^0$ 0.8 $\frac{m}{s} m$ 80 $kg \Delta T$ 0.1 sPerformance evaluation simulation constants~~

Two scenarios were developed for this evaluation. Figure 14 ~~displays~~ shows their physical topology. ~~On~~ In these figures, black regions represent walls, green ~~represents pedestrian spawners ,~~ ~~meaning~~ regions denote pedestrian spawners (locations where pedestrians are added ~~with a certain frequency~~at a specific frequency), and red ~~represents destinations, meaning that pedestrians are trying to go there, and they get removed from the simulation once they have arrived. Spawners and destinations are marked with a letter to represent their pairing. So~~ regions represent destinations, where pedestrians are removed upon arrival. Each spawner is paired with a destination, indicated by letters. For example, pedestrians from spawner A ~~will attempt to get to~~ aim for destination A.



(a) Scenario 1 - Bottleneck

(b) Scenario 2 - Corridor

Figure 14 – Performance scenario topologies

Both ~~of the developed scenarios have a simple topology and small surface area. However~~developed scenarios feature simple topologies and relatively small surface areas. Nevertheless, they were ~~created with a basis on actual topologies~~ designed based on real-world topologies commonly used during the development of novel pedestrian models. ~~Similar topologies to the ones seen in~~ Scenario 1 ~~are oftentimes used to test whether models display~~ reflects setups typically employed to evaluate the "Faster-Is-Slower Effect"~~, where an increase in pedestrian speed may actually~~ ," in which increasing pedestrian speed can paradoxically reduce total throughput when ~~there is a bottleneck~~

~~on the path~~a bottleneck is present. Scenario 2, in contrast, focuses on lane formation, a ~~situation where pedestrians create lanes of traffic when there are many pedestrians~~ phenomenon in which pedestrians naturally organize into directional lanes when multiple streams moving in different directions intersect.

~~Each scenario was measured in terms of how long~~ For these experiments, the frequency with which pedestrians are added to the simulation, denoted as f_p and measured in $pedestrians/s$, was chosen as the main independent variable. This allows for the measurement of the performance of the simulators as the number of pedestrians, and therefore interactions, increases. For example, with $f_p = 0.5$, a pedestrian is added every other second, while $f_p = 8$ means that a pedestrian is added every 0.125 seconds. Agents are placed randomly inside their insertion regions and all have the objective of reaching their pre-assigned destination.

The dependent variable being measured during these tests was the total execution time. It was chosen because, for most scientific use cases, the total time it takes to execute a scenario is the most important property that is performance dependent. Other related metrics, such as mean time per simulation step, are not easily measurable in Jupedsim but can be derived from the mean total time. Each combination of scenario and f_p was simulated 50 times, and the mean of the execution time was recorded.

All experiments use the Social Forces Model, with simulator constants standardized as shown in Table 4. These include the calibration constants from Equation 3, Equation 5, and Equation 6. Since the algorithms used to compute the shortest path from a pedestrian to its destination, a key component of the motivation force, are substantially different and incompatible, their values were left at their default settings. The total amount of time simulated, denoted as T_s, was standardized to a value of 5 minutes, meaning that each run would simulate the equivalent of 5 minutes ~~in simulation time under various circumstances. This amount of time~~ and then stop. This duration was chosen because it ~~is enough for the~~ allows low and medium f_p values to reach equilibrium~~(when the number of pedestrians entering the simulation and leaving is the same), but not enough for the higher values to completely fill the whole simulation areawith pedestrians at the higher values. Each scenario was executed 10 times and an average was taken as the result~~, where the inflow and outflow of pedestrians are balanced, while preventing higher f_p values from completely saturating the simulation area. Simulations were executed on standardized hardware, with key specifications listed in Table 5.

5.3 ~~Qualitative Evaluation Results~~

5.3 Qualitative evaluation results

Following the directives ~~defined~~ outlined in Section 5.1, each ~~scenario was implemented sequentially. In cases where there were~~ use case was implemented in sequence. Whenever examples, extension points~~or documentation pointing towards~~, or documentation provided by the developers suggested a recommended implementation ~~path from the developers, these~~ approach, these guidelines were followed. ~~When there was no explicit guidance, various approaches were attempted to arrive at the best possible~~ In the absence of such explicit directions, multiple alternative solutions were explored to determine the most suitable one according to the ~~evaluation criteria defined. Below is a description of the results with some additional commentary~~defined evaluation criteria. ~~A summary of the same can be seen in .~~ The results of this process, along with additional commentary, are presented below.

Parameter	Value	Unit
A_i	2000	N
B_i	0.08	m
k	120000	$\frac{kg}{s^2}$
κ	240000	$\frac{kg}{m \cdot s}$
r_i	0.3	m
τ	0.5	s
v_i^0	0.8	$\frac{m}{s}$
m	80	kg
ΔT	0.1	s
T_s	5	min

Table 4 – Controlled variables for the performance evaluation

Property	Value
Operating System	Linux Mint 22.1 Cinnamon
Linux Kernel	6.8.0-71-generic
CPU	AMD Ryzen 5 5560U
Graphics Card	AMD Radeon Vega Mobile Series
RAM	12 GB

Table 5 – Simulation hardware and software

Each simulator had very similar results in ~~scenarios~~ Use Cases 1, 2, and 3. These use cases are not complicated ~~setups and serve as a comparison basis for the other ones. However~~and serve primarily as a baseline for comparison with the more complex ones. Nonetheless, they already reveal ~~some trends that will persist over~~ several trends that persist throughout most of the ~~other scenarios~~remaining use cases. Firstly, even in ~~basic scenarios~~these basic use cases, ECSMoS ~~is more complicated than desired. This is due to how~~ exhibits a higher level of complexity than desired, largely due to the open-ended ~~its architecture is. Because~~ nature of its architecture. Since new systems can be ~~added to any particular point in the schedule~~inserted at any point within the simulation schedule, understanding the overall control flow often becomes challenging in practice. On the other hand, ~~in practice it means that it is always somewhat complicated to understand the control flow of the simulator. However,~~ implementing model variations and hybrid models ~~works well, as these~~ is simple: such extensions can be added on top of existing models ~~easily via the addition of~~ by adding a few systems that work together with the ~~older~~ pre-existing ones.

Secondly, JuPedSim generally performs worse than the others being studied under these criteria. Such a pattern stems from its highly coupled internal structure. Even in basic ~~scenarios, JuPedSim requires many changes~~ use cases, JuPedSim necessitates numerous modifications all around the codebase in the form of string comparisons and ~~others~~ other conditional mechanisms that define how execution will go depending on each model. This issue is most notable in the interface between Python and C++ code, where there are large sections of model-specific code that are executed depending on a sequence of chained conditions.

Finally, Vadere receives a ~~downgrade in~~ lower reusability scores starting on ~~scenario~~ Use Case 2 due to the fact that its ~~pre-existing~~ existing model implementations have a somewhat limited level

Simulator / Use Case	ECSMoS			Vadere			JuPedSim		
	Implementability	Reusability	Complexity	Implementability	Reusability	Complexity	Implementability	Reusability	Complexity
(1) New Model	Good	Good	Fair	Good	Good	Good	Fair	Fair	Fair
(2) Model variation	Good	Good	Fair	Good	Fair	Good	Fair	Fair	Fair
(3) Hybrid model	Good	Good	Fair	Good	Fair	Good	Fair	Fair	Fair
(4) Procedural environment changes	Good	Good	Fair	Fair	Fair	Poor	Poor	Poor	Poor
(5) New pathfinding algorithm	Good	Fair	Fair	Poor	Fair	Poor	Fair	Fair	Fair
(6) Queuing and loitering	Good	Good	Fair	Good	Poor	Good	Good	Good	Good

Table 6 – Summary of qualitative results.

of extensibility. With better choices when it comes to how the code is divided or how classes and interfaces are structured, it could cut down on the amount of code duplication significantly.

~~Scenario~~ Use Case 4 introduces ~~more~~ additional challenges when it comes to flexibility. Both Vadere and JuPedSim ~~suffer~~ encounter difficulties on this front, as they assume static environments~~, which~~ . This is a reasonable expectation for some simpler situations, but that is not a universal property. Within Vadere, it is still possible to accommodate these with some work, but it will require reengineering some portions of the simulator itself. ~~However, they are basically unimplementable in JuPedSim~~ In contrast, implementing these features in JuPedSim is practically unfeasible, as it would require reengineering most of the simulator. In comparison, ECSMoS handles such things much better. Bevy ECS provides methods for detecting this kind of changes and, in many situations, these can be added with little to no modification to the code of the systems themselves. In the situation where changes to the systems are necessary, those are limited and much more predictable.

~~(1) New Model Good Good Fair Good Good Good Fair Fair Fair(2) Model variation Good Good Fair Good Fair Good Fair Fair Fair(3) Hybrid model Good Good Fair Good Fair Good Fair Fair Fair (4) Procedural environment changes Good Good Fair Fair Fair Poor Poor Poor Poor (~~Use Case 5 ~~) Implementing new pathfinding algorithm Good Fair Fair Poor Fair Poor Fair Fair Fair (6) Queuing and loiteringGood Good Fair Good Poor Good Good Good Good Summary of qualitative results.~~

~~Scenario 5~~ causes a reversal in the grades between Vadere and JuPedSim. Vadere's pathfinding algorithm, ~~referred to~~ known as floor fields, is much more integrated with the simulator's architecture and existing models~~than JuPedSim's, which uses~~ , whereas JuPedSim relies on triangulation to determine the best possible path for each pedestrian. ~~While it is possible to replace floor fields with some other~~ Although floor fields could theoretically be replaced with an alternative mechanism, it would ~~involve changing much of~~ require extensive modifications to both the simulator code and all ~~current implementationsof models~~existing model implementations, making it impractical. In ~~, such a thing is perfectly possible. Reuse of code~~ contrast, ECSMoS supports such changes seamlessly. While code reuse between the current pathfinding system and a new one may be limited, ~~but the architecture will not impose additional challenges and the changes necessary to allow already~~ the architecture itself does not introduce additional obstacles, and the modifications required to enable existing models to ~~use it are small and concentrated in singular places~~utilize the new system are minimal and localized.

Finally, in ~~scenario~~ Use Case 6, JuPedSim performs very well. Its architecture ~~actively expects such behavior, so it is well integrated and quite flexibleto changes. While~~ is specifically designed to accommodate this type of behavior, making it both well-integrated and highly flexible. Although ECSMoS was not ~~designed with this in mind, this can be accomplished similarly to how hybrid modelswere implemented in scenario 3. Queuing and loitering work as an extension~~ explicitly designed for this use case, similar functionality can be achieved using the same approach as in Use Case 3 for hybrid models, where queuing and loitering are implemented as extensions on top of the already available model. ~~Vadere's implementation in theorycould be done in the same way as~~ In theory, Vadere could adopt a similar approach to ECSMoS~~'s~~, but in practice~~it~~, this would require rewriting the ~~whole model to take this into account due to some limits to~~ entire model to account for these behaviors due to inherent limitations in its extensibility. Therefore, ~~it gets a downgrade~~ Vadere receives a lower score in the reusability criteria.

As the results in Table 6 show, Vadere and ECSMoS are the best for basic ~~scenarios~~use cases, with Vadere ~~having a slight edge~~holding a slight advantage. Each one has its strong and weak points, but ~~they are generally good choices~~ both are generally suitable for implementing a novel pedestrian model that does not ~~take into account more complicated things such as changes to the environment~~involve complex dynamics, such as environmental changes. However, ~~once more involved changes are necessary~~when use cases require more sophisticated modifications, ECSMoS ~~pulls ahead because of its~~ demonstrates a clear advantage due to its superior flexibility. In ~~contrast,~~

~~JuPedSim has generally worse results than these two~~comparison, JuPedSim generally yields worse results across the use cases, with its only standout result ~~on scenario 6.~~ in Use Case

5.4 Performance ~~Evaluation Results~~evaluation results

Similarly to the qualitative results, the performance results were very ~~similar~~ close for the less demanding ~~cases~~scenarios, while diverging significantly on later ones. Figure 15 shows ~~us~~ the full results in detail. In ~~that figure~~the graph, scenarios are identifiable by the shape of the markers~~used on the graph~~, while each simulator ~~got its own~~ is represented by a distinct color. The shaded areas ~~around~~ surrounding each trend line ~~are~~ indicate the standard deviation of ~~each measurement, which measures~~ the measurements, reflecting the variability of ~~each result~~the results.

As ~~it is possible to see, the results~~ can be observed, the performance across simulators and scenarios is very similar for lower values of f_p~~are very similar across simulators and scenarios~~. With such low ~~amounts of pedestrians~~pedestrian densities, the base costs of loading data into memory, pre-processing, ~~post-processing and other fixed costs dominate . There is still some variation~~ and post-processing dominate the overall runtime. Some variation remains, most notably from Vadere, which is written in Java and therefore ~~has a slower general performancethan the other ones because~~ exhibits comparatively slower performance, as Java is not fully compiled ~~. Instead it is~~ but instead partially compiled into an ~~intermediary language that then needs to be interpreted . JuPedSim~~ intermediate language that must be interpreted at runtime. JuPedSim, which uses Python, ~~a fully~~ an interpreted language, ~~but most of its core is written~~ has higher performance because the core computations are implemented in C++~~and the Python wrapper simply serves to give~~. Python is mostly used as a wrapper for a convenient interface ~~to the user, as it passes execution~~ that delegates execution directly to the C++ layer~~immediately~~.

Starting at $f_p = 3$, no data is available for JuPedSim ~~on scenario 2. At~~ in Scenario 2, and by $f_p = 6$, the same ~~becomes true for scenario~~ limitation occurs for Scenario 1. This ~~is due to~~ issue comes from a limitation in JuPedSim. As pedestrian densities increase inside the simulation, it is possible ~~in rare circumstances~~ for the repulsive interaction between pedestrians to ~~be so strong that a pedestrian may be going so fast~~ become too strong in certain circumstances. These strong forces can cause an individual to move at such a high speed that its position in the next simulation loop goes over the barriers that limit the simulation area, ~~so it appears on the outside. Said~~ causing the agent to appear outside the simulation area. This behavior is a common ~~error for~~ issue in both simulators and games, and stems from a simplification ~~of~~ in the collision detection ~~code. Instead of checking if the pedestrians would have to go through a wallto arrive at a certain position~~algorithm. Rather than verifying whether a pedestrian's path would intersect a wall, the code simply checks ~~if there is~~ for the presence of a wall at the location the agent ~~is going to~~will be in the next simulation step. At low speeds this approach is sufficient, but it has limited applicability at higher speeds.

Such high speeds are not usually possible for pedestrians in the real world, but similar situations can arise in many models, including the Social Forces Model. A common location for this to happen is ~~around the~~ near regions where pedestrians are added. In these locations, a new pedestrian may be added much closer to others than ~~it would be normally~~ would normally be possible, creating ~~large~~ unusually large repulsive forces. Both Vadere and ECSMoS ~~handle it by using~~ mitigate this issue by enforcing a configurable maximum speed~~that is~~, set well below the threshold ~~where it could cause~~ that could lead to such problems. Since such speeds are physically impossible in real-world scenarios, ~~a maximum speed helps the model behave in a more realistic way~~the maximum speed constraint helps to ensure the model behaves more realistically. However, JuPedSim ~~does not have that check nor does it provide a way~~ lacks this safeguard and any means to configure it. As a result, even ~~modest densities can overwhelm it~~ moderate pedestrian densities can cause failures in certain situations. ~~It~~

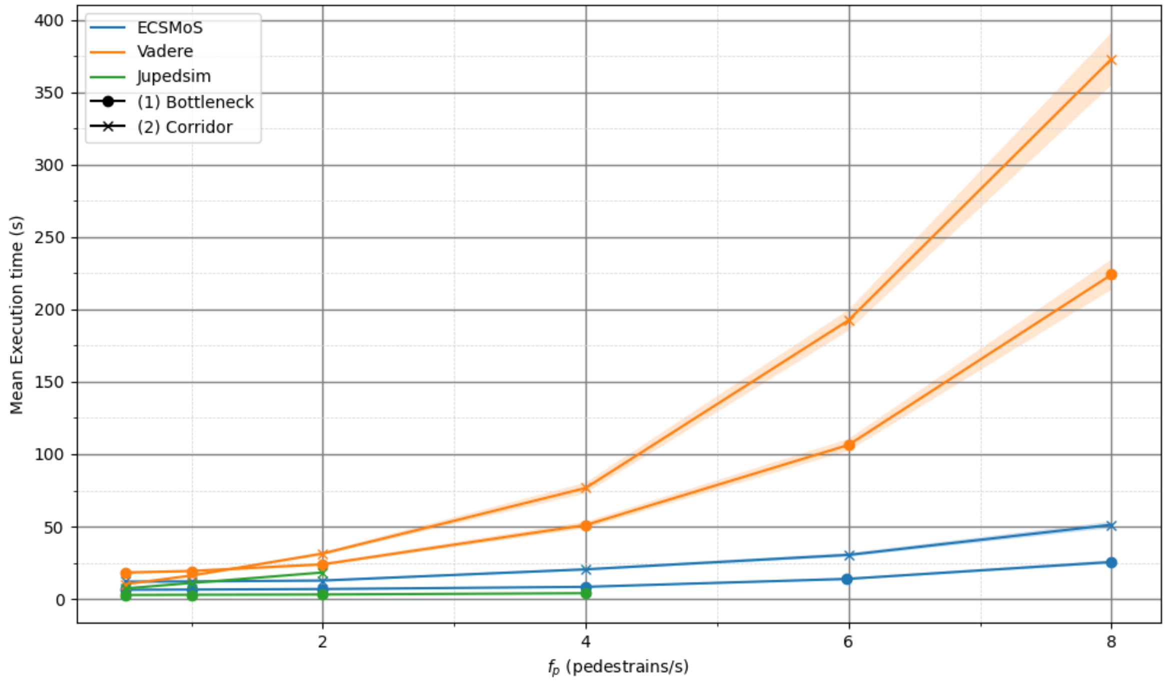


Figure 15 – Summary of performance results

~~does not happen on all runs, but they become common~~ While this does not occur in every simulation run, it becomes frequent enough at $f_p = 3$ and $f_p = 6$ ~~that it becomes infeasible to gather~~ to make it infeasible to obtain representative results.

Vadere and ECSMoS exhibit an overall quadratic growth in simulation duration as ~~the frequency increases. Although there are many ways to optimize it,~~ f_p increases. While various optimization strategies exist, this behavior is an inherent consequence of the Social Forces Model~~naturally leads to that kind of growth~~. Each pedestrian both applies forces to others and ~~receives a reciprocal force~~ experiences reciprocal forces from them. These ~~need to~~ must be computed for ~~each individual~~ every pair of agents, ~~which grows with the square of the~~ leading to a computation cost that grows quadratically with the number of agents in ~~a~~ the simulation. However, at higher frequencies , ~~there is~~ a large difference in performance emerges between the two~~, but it is difficult~~. Still, it is challenging to quantify how much of this ~~comes from the change in architecture and how much comes from the difference in~~ difference is attributable to architectural design versus the underlying programming language.

~~The comparing~~

Comparing ECSMoS to JuPedSim is simpler ~~in that aspect~~when it comes to the performance of their respective programming languages, as C++ and Rust have very similar performance profiles [49]. In the proposed scenarios, both simulators have ~~near identical performance for the~~ nearly identical performance at lower frequencies. This ~~could indicate some degree of equivalence between their architectures~~ suggests that their architectures may be roughly equivalent in terms of performance~~, but~~. However, due to the lack of data for the later runs for JuPedSim~~means that~~, it is not possible to make reliable ~~assertions on this matter~~conclusions about their relative performance in more demanding scenarios.

~~Summary of performance results~~

6. Conclusion

This research aimed to determine the benefits and drawbacks of introducing the ECS architecture into the realm of scientific pedestrian simulations with a focus on flexibility. As part of this, the area of pedestrian simulations was studied, including currently available pedestrian models and a survey of pedestrian simulators. With this context established, the characteristics of the ECS architecture were laid out in comparison to the ones used by current simulators. Finally, a novel simulation framework, ECSMoS, was proposed, developed and compared against some of the most popular pedestrian simulators, Vadere and JuPedSim.

The results of the qualitative analysis show that ECSMoS is more flexible when compared to the other subjects of study overall, with special highlights in situations that require procedural environment changes, where both of the other simulators face significant challenges. However, this comes at the expense of complexity. In fact, ECSMoS is generally more complex than the alternatives in simpler scenarios. ~~This drawback only ever disappears~~ However, this drawback becomes less relevant when finer control or flexibility is required. In terms of performance, ECSMoS is just as or more performant than the alternatives in all studied scenarios. Still, these results are not fully comparable, as there are other significant factors that may affect performance that cannot be isolated.

Based on these results, it is possible to conclude that ECS is a viable alternative to currently accepted programming paradigms used in the simulation of pedestrians. Its benefits come in the form of flexibility and control, but they come with drawbacks in terms of required learning for unfamiliar researchers and a generally complex execution flow and debugging procedures. Therefore, it is best used for implementing models that require features that are not currently standard or break the general assumptions when it comes to pedestrian model structure.

Future works in ~~this area~~ exploring the limits of the ECS architecture should focus on expanding the current simulation core with features necessary for a fully fledged simulator. These may include a better UI, mechanisms for defining scenarios via external files, better data gathering, supporting other export formats, and many others. Additionally, it may be possible to use some of the strategies and concepts of ECS on other simulators. This path of research may allow researchers to have the benefits of ECS without needing to ~~learn a novel architecture or reimplement existing tools~~go through the sharp learning curve associated with ECS and Rust.

References

[1] Juliane Adrian, Armin Seyfried, and Anna Sieben. “Crowds in Front of Bottlenecks at Entrances from the Perspective of Physics and Social Psychology”. In: *Journal of The Royal Society Interface* 17.165 (Apr. 2020), p. 20190871. DOI: 10.1098/rsif.2019.0871. (Visited on 10/14/2025).

[2] Cornelia von Krüchten and Andreas Schadschneider. “Empirical Study on Social Groups in Pedestrian Evacuation Dynamics”. In: *Physica A: Statistical Mechanics and its Applications* 475 (June 2017), pp. 129–141. ISSN: 0378-4371. DOI: 10.1016/j.physa.2017.02.004. (Visited on 10/20/2024).

[3] Mahsa Golchoubian et al. “Pedestrian Trajectory Prediction in Pedestrian-Vehicle Mixed Environments: A Systematic Review”. In: *IEEE Transactions on Intelligent Transportation Systems* 24.11 (2023), pp. 11544–11567. DOI: 10.1109/TITS.2023.3291196.

[4] Haoyang Liang et al. “Unraveling the Causes of the Seoul Halloween Crowd-Crush Disaster”. In: *PLOS ONE* 19.7 (July 12, 2024), e0306764. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0306764. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0306764 (visited on 10/20/2024).

[5] Dirk Helbing, Illés Farkas, and Tamás Vicsek. “Simulating Dynamic Features of Escape Panic”. In: *Nature* 407 (Sept. 28, 2000), pp. 487–490. DOI: 10.1038/35035023.

[6] George Kouskoulis and Constantinos Antoniou. “Systematic Review of Pedestrian Simulation Models with a Focus on Emergency Situations”. In: *Transportation Research Record* 2604.1 (Jan. 2017), pp. 111–119. ISSN: 0361-1981. DOI: 10.3141/2604-14. (Visited on 10/14/2025).

[7] Dermot Barr et al. “Beyond ‘Stampedes’: Towards a New Psychology of Crowd Crush Disasters”. In: *British Journal of Social Psychology* 63.1 (2024), pp. 52–69. ISSN: 2044-8309. DOI: 10.1111/bjso.12666. (Visited on 10/14/2025).

[8] Giovanni Tesoriere et al. “Modelling and Simulation of Passenger Flow Distribution at Terminal of Catania Airport”. In: *AIP Conference Proceedings* 2040.1 (Nov. 2018), p. 140006. ISSN: 0094-243X. DOI: 10.1063/1.5079195. (Visited on 10/14/2025).

[9] Hairong Dong et al. “State-of-the-Art Pedestrian and Evacuation Dynamics”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.5 (2020), pp. 1849–1866. DOI: 10.1109/TITS.2019.2915014.

[10] Benedikt Kleinmeier et al. “Vadere: An Open-Source Simulation Framework to Promote Interdisciplinary Understanding”. In: *Collective Dynamics* 4 (Sept. 3, 2019), pp. 1–34. ISSN: 2366-8539. DOI: 10.17815/CD.2019.21. URL: https://collective-dynamics.eu/index.php/cod/article/view/A21 (visited on 11/08/2024).

[11] Andrea Gorrini, Giuseppe Vizzari, and Stefania Bandini. “Age and Group-driven Pedestrian Behaviour: From Observations to Simulations”. In: *Collective Dynamics* 1 (July 21, 2016), pp. 1–16. ISSN: 2366-8539. DOI: 10.17815/CD.2016.3. URL: https://www.collective-dynamics.eu/index.php/cod/article/view/A3 (visited on 10/14/2025).

[12] Libi Fu et al. “An Experimental Study on Bidirectional Pedestrian Flow Involving Individuals with Simulated Disabilities in a Corridor”. In: *Safety Science* 150 (June 2022), p. 105723. ISSN: 0925-7535. DOI: 10.1016/j.ssci.2022.105723. (Visited on 10/14/2025).

[13] Mohammed Alrashed and Jeff Shamma. “Agent Based Modelling and Simulation of Pedestrian Crowds in Panic Situations”. In: *Collective Dynamics* 5 (Aug. 2020), pp. 463–466. DOI: 10.17815/CD.2020.100. URL: https://collective-dynamics.eu/index.php/cod/article/view/A100.

[14] Laurent Voisard et al. “A Mapping Study of the Entity Component System Pattern”. In: *2025 IEEE/ACM 9th International Workshop on Games and Software Engineering (GAS)*. 2025, pp. 33–40. DOI: 10.1109/GAS66647.2025.00010.

[15] Lars I. Hatledal et al. “Vico: An entity-component-system based co-simulation framework”. In: *Simulation Modelling Practice and Theory* 108 (2021), p. 102243. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2020.102243. URL: https://www.sciencedirect.com/science/article/pii/S1569190X20301726.

[16] Brennen T. Garland. “Designing and Building a Radar Simulation Using the Entity Component System”. PhD thesis. Air Force Institute Of Technology, 2021.

[17] Francisco Martinez-Gil et al. “Modeling, Evaluation, and Scale on Artificial Pedestrians: A Literature Review”. In: *ACM Comput. Surv.* 50.5 (Sept. 2017), 72:1–72:35. ISSN: 0360-0300. DOI: 10.1145/3117808. (Visited on 05/17/2025).

[18] Keke Huang et al. “Behavior-based cellular automaton model for pedestrian dynamics”. In: *Applied Mathematics and Computation* 292 (2017), pp. 417–424. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2016.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0096300316304325.

[19] Dirk Helbing and Peter Molnar. “Social Force Model for Pedestrian Dynamics”. In: *Physical Review E* 51.5 (May 1, 1995), pp. 4282–4286. ISSN: 1063-651X, 1095-3787. DOI: 10.1103/PhysRevE.51.4282. arXiv: cond-mat/9805244. URL: http://arxiv.org/abs/cond-mat/9805244 (visited on 05/17/2025).

[20] Claudio Feliciani and Katsuhiro Nishinari. “Empirical analysis of the lane formation process in bidirectional pedestrian flow”. In: *Phys. Rev. E* 94 (3 Sept. 2016), p. 032304. DOI: 10.1103/PhysRevE.94.032304. URL: https://link.aps.org/doi/10.1103/PhysRevE.94.032304.

[21] *Pedestrian simulation software | PTV Viswalk | PTV Group*. https://www.ptvgroup.com/en/products/pedestrian-simulation-software-ptv-viswalk. Accessed: 2025-09-27.

[22] *Oasys | MassMotion*. https://www.oasys-software.com/products/massmotion/. Accessed: 2025-09-27.

[23] *LEGION | Bentley Systems | Infrastructure Engineering Software Company*. https://br.bentley.com/software/legion/. Accessed: 2025-09-27.

[24] *AnyLogic: Simulation Modeling Software Tools & Solutions*. https://www.anylogic.com/. Accessed: 2025-09-27.

[25] Sylvain Faure. *Cromosim: Crowd Motion Simulation*. Version 2.1.0. Dec. 30, 2024. URL: https://github.com/sylvain-faure/cromosim.

[26] Timo Korhonen and Simo Hostikka. *Fire Dynamics Simulator with Evacuation: FDS+Evac: Technical Reference and User's Guide*. VTT Working Papers. Espoo: VTT Technical Research Centre of Finland, 2009.

[27] Axel Meinert et al. "Simulation von Menschenmengen Im Urbanen Umfeld Mit OpenStreetMap-Daten". In: *gis. Science* (2019).

[28] Armel Ulrich Kemloh Wagoum et al. "JuPedSim: An Open Framework for Simulating and Analyzing the Dynamics of Pedestrians". In: Dec. 2015.

[29] Sean Curtis, Andrew Best, and Dinesh Manocha. "Menge: A Modular Framework for Simulating Crowd Movement". In: *Collective Dynamics* 1 (Mar. 15, 2016), pp. 1–40. ISSN: 2366-8539. DOI: `10.17815/CD.2016.1`. URL: `https://collective-dynamics.eu/index.php/cod/article/view/A1` (visited on 11/08/2024).

[30] Michael J. Seitz et al. "The Superposition Principle: A Conceptual Perspective on Pedestrian Stream Simulations". In: *Collective Dynamics* 1 (Mar. 15, 2016), pp. 1–19. ISSN: 2366-8539. DOI: `10.17815/CD.2016.2`. URL: `https://collective-dynamics.eu/index.php/cod/article/view/A2` (visited on 11/26/2024).

[31] Peter M. Kielar, Daniel H. Biedermann, and André Borrmann. "MomenTUMv2: A Modular, Extensible, and Generic Agent-Based Pedestrian Behavior Simulation Framework". In: (Jan. 2016).

[32] Pablo Alvarez Lopez et al. "Microscopic Traffic Simulation using SUMO". In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL: `https://elib.dlr.de/124092/`.

[33] Ewout ter Hoeven et al. "Mesa 3: Agent-based Modeling with Python in 2025". In: *Journal of Open Source Software* 10.107 (Mar. 2025), p. 7668. ISSN: 2475-9066. DOI: `10.21105/joss.07668`. (Visited on 07/21/2025).

[34] George Datseris, Ali R. Vahdati, and Timothy C. DuBois. "Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity". In: *SIMULATION* 0.0 (Jan. 2022), p. 003754972110688. DOI: `10.1177/00375497211068820`. URL: `https://doi.org/10.1177/00375497211068820`.

[35] *Specs and Legion, two very different approaches to ECS*. `https://csherratt.github.io/blog/posts/specs-and-legion/`. Accessed: 2025-09-27.

[36] SanderMertens. *Flecs*. Version 4.0.5. Mar. 18, 2025. URL: `https://www.flecs.dev/flecs/`.

[37] Unity Technologies. *DOTS - Unity's Data-Oriented Technology Stack*. Version 6.1. June 10, 2025. URL: `https://unity.com/dots`.

[38] Bevy Engine. *Bevy*. Version 0.16.0. June 24, 2025. URL: `https://bevyengine.org/`.

[39] Amethyst Foundation. *Legion*. Version 0.4.0. Feb. 25, 2021. URL: `https://github.com/amethyst/legion`.

[40] Benjamin Saunders. *hecs*. Version 0.10.5. May 5, 2024. URL: `https://github.com/Ralith/hecs`.

[41] Michele Caini. *EnTT*. Version 3.15.0. Mar. 19, 2025. URL: `https://github.com/skypjack/entt`.

[42] Michele Caini. *Dylan Ancel*. Version 0.8. Mar. 31, 2025. URL: https://github.com/leudz/shipyard.

[43] Alec Thomas. *EntityX - A fast, type-safe C++ Entity Component System*. Version 1.1.2. Apr. 25, 2025. URL: https://github.com/alecthomas/entityx.

[44] Amethyst Foundation. *Specs Parallel ECS*. Version 0.20.0. Sept. 25, 2023. URL: https://amethyst.github.io/specs/.

[45] Vladyslav Batyrenko. *Bevy egui*. Version 0.36.0. Sept. 13, 2025. URL: https://github.com/vladbat00/bevy_egui.

[46] Melissa De Iuliis et al. "Large scale simulation of pedestrian seismic evacuation including panic behavior". In: *Sustainable Cities and Society* 94 (2023), p. 104527. ISSN: 2210-6707. DOI: https://doi.org/10.1016/j.scs.2023.104527. URL: https://www.sciencedirect.com/science/article/pii/S2210670723001385.

[47] Inhi Kim, Ronald Galiza, and Luis Ferreira. "Modeling pedestrian queuing using microsimulation". In: *Transportation Research Part A: Policy and Practice* 49 (2013), pp. 232–240. ISSN: 0965-8564. DOI: https://doi.org/10.1016/j.tra.2013.01.018. URL: https://www.sciencedirect.com/science/article/pii/S0965856413000256.

[48] MD Jahedul Alam, Muhammad Ahsanul Habib, and Devin Holmes. "Pedestrian movement simulation for an airport considering social distancing strategy". In: *Transportation Research Interdisciplinary Perspectives* 13 (2022), p. 100527. ISSN: 2590-1982. DOI: https://doi.org/10.1016/j.trip.2021.100527. URL: https://www.sciencedirect.com/science/article/pii/S2590198221002323.

[49] Ignas Plauska, Agnius Liutkevičius, and Audronė Janavičiūtė. "Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller". In: *Electronics* 12.1 (2023). DOI: 10.3390/electronics12010143. URL: https://www.mdpi.com/2079-9292/12/1/143.