Implementation of cellular automata using behavior-driven development

Implementación de autómatas celulares usando desarrollo dirigido por comportamiento

VictorVargas-Forero[1]
Luz Muñoz - Ceballos[2]
Angel Garcia - Baños[3]

[1]Universidad del Valle (Colombia). Email: victor.m.vargas@correounivalle.edu.co; orcid:http://orcid.org/0000-0002-2317-7365

[2]Universidad del Valle (Colombia). Email: luz.estela.munoz@correounivalle.edu.co; orcid: http://orcid.org/0000-0002-2675-1238

[3]Universidad del Valle (Colombia). Email: angel.garcia@correounivalle.edu.co; orcid: http://orcid.org/0000-0002-5388-6912

Received: 15-11-2018 Accepted: 06-08-2019

How to cite: Vargas-Forero, Victor; Muñoz-Ceballos, Luz; García-Baños, Ángel (2020). Implementación de autómatas celulares usando desarrollo dirigido por comportamiento . *Informador Técnico*, 84(1), 48-66.
https://doi.org/10.23850/22565035.2257

Resumen

Este trabajo tuvo como objetivo presentar parte de la implementación de un autómata celular usando el desarrollo dirigido por comporta-miento (BDD-behavior-driven development) como estrategia para la construcción de una herramienta que permita modelar la accesibilidad peatonal dentro de un espacio abierto como es el campus de la Universidad del Valle en Cali, Colombia. Para la implementación se usó BDD como método de desarrollo, Python como lenguaje de programación y Behave para escribir las pruebas en lenguaje natural. Como resultados, se presentaron parte del modelo implementado del autómata celular, algunos de los requerimientos con sus respectivos escenarios para BDD e implementación, donde se explicó el uso del framework Behave en Python. Se concluyó que BDD es una buena herramienta para este tipo de proyectos porque se obtuvo como resultado 706 líneas de código fuente correspondientes al modelo del autómata celular probadas con un total de 144 pruebas definidas desde la fase inicial del proyecto y superadas satisfactoriamente, una encuesta de aceptación del uso de la metodología BDD por parte de los usuarios del proyecto y primera aplicación de BDD en una simulación de movilidad peatonal.

Palabras clave: software ágil; BDD, autómatas celulares; accesibilidad peatonal, procesos y métodos de desarrollo de software.

Abstract

In this work, part of the implementation of a cellular automaton is presented using Behavior-Driven Development (BDD) as a strategy for the construction of a tool that allows modeling pedestrian accessibility within an open space, such as the campus of the Universidad del Valle in Cali, Colombia. For the implementation, BDD was used as the development method, 'Python' as the programming language, and 'Behave' to write the tests in natural language. As results, part of the implemented model of the cellular automaton and some of the requirements with their respective scenarios for BDD and implementation were presented, where the use of the Behave framework in Python was explained. It was concluded that BDD is a good tool for this type of project because the result obtained was 706 lines

of source code corresponding to the cellular automaton model, testted with a total of 144 tests defined from the initial phase of the project and satisfactorily passed, according to an acceptance survey on the use of the BDD methodology by the project's users, and the first application of BDD in a simulation of pedestrian mobility.

Keywords: agile software; BDD; cellular automata; pedestrian accessibility; software development processes and methods.

1. Introduction

The experience of an implementation of the model of a cellular automaton is presented to analyze the pedestrian accessibility inside the university campus of the Meléndez citadel of the Universidad del Valle, taking as a starting point the study carried out by Buitrago and Kattán (2011), in addition to implementing the application directed by *Behavior-driven development* (BDD) (North, 2006), as a methodological tool for said construction, based on the result of the undergraduate work by Muñoz (2017).

Using the cellular automaton, the core of the university campus pedestrian simulator was built, a model that was presented in Vargas-Forero; Muñoz-Ceballos; García-Baños; Jaramillo-Molina (2019). There are several studies that have focused on microsimulation problems, for example (Gipps; Marksjö, 1985), but most of these problems focus on the evacuation of pedestrians (Gudowski; Was, 2007) and some involve environmental characteristics and the behaviors of the neighbors (Was, 2005). The problems of accessibility and pedestrian mobility are currently a highly analyzed field, as studied in Martins-Gonçalves; Nuñez-Basrantes (2017); Chen *et al*. (2018); Qin; Curtin; Rice (2018).

On the other hand, another challenge for the working group project was testing agile software development in a research project using cellular automata. Since 2001, with the initiative of the Agile Manifesto (Beck, 2001), a very strong trend that seeks to change the way of developing software began (traditional methodologies, the most representative for the RUP era (Khamis; Abdelmonem, 2002). As it can be seen in Poppendieck and Cusumano (2012), there is a great difference between agile development processes and traditional methodologies; for this reason, a set of principles is proposed to guide the development process. From the Agile Manifesto, many initiatives arise such as Test-Driven Development (TDD), Acceptance Test Driven Development (ATDD), Behavior Driven Development (BDD), among others. It was decided to work with BDD due to the characteristics presented by this software development process, such as ubiquitous language (based on the domain of the problem), iterative decomposition process, text description with user stories and scenario templates, automated acceptance tests with mapping rules, readable behavior-oriented specification code, and different behavior-driven development phases. In addition, Behave (Engel; Rice; Jones, 2012) was chosen as a support tool for BDD, as it is one of the tools that fulfills the most requirements (Solis; Wang, 2011).

Therefore, this research is developed as follows: in the first part, the proposed methodology is introduced. In the second part, a brief presentation of the model of the cellular automata to be built is made. The third one shows some of the results obtained when building the model using BDD and at the end, in the fourth part, the conclusions of the carried-out development are presented.

2. Methodology

The work was developed following the *BDD* guidelines (North, 2006; Ferguson-Smart, 2014), which focus on the expected "behaviors" of the software and not on how its logic should be implemented, facilitating understanding between the final user and the developer. This characteristic was desirable in the project because it was expected that the requirements would change very frequently throughout the execution of the project. BDD arises from the Agile Manifesto initiative (Beck *et al.*, 2001), which is based on the values indicated in Table 1, giving greater value to the elements on the left than those on the right:

Table 1.
Agile Manifesto BDD

Left		Right
Individuals and Interactions	about	Processes and Tools
Working Software	about	Extensive Documentation
Collaboration with the client	about	Contract Negotiation
Change response	about	Following a Plan

Source: own elaboration.

From this initiative, there are others that seek to improve the traditional methodologies that were used to date. One of these is TDD (Jurado, 2010), which involves two other practices: writing the tests before writing the program code and refactoring until finishing to build the whole solution. BDD can be seen as an enhancement to TDD, where the features of TDD are incorporated into BDD, so that TDD focuses on unit testing and BDD focuses on higher-level testing, functionality, acceptance, the search to define the problem and not how it should be solved.

In Figure 1 it is observed that the methodological strategy in general for BDD, consists of building user acceptance tests, unit tests, then making the necessary code to pass said tests and refactoring with the new tests, carrying out this process in cycles and incorporating the new functionalities.

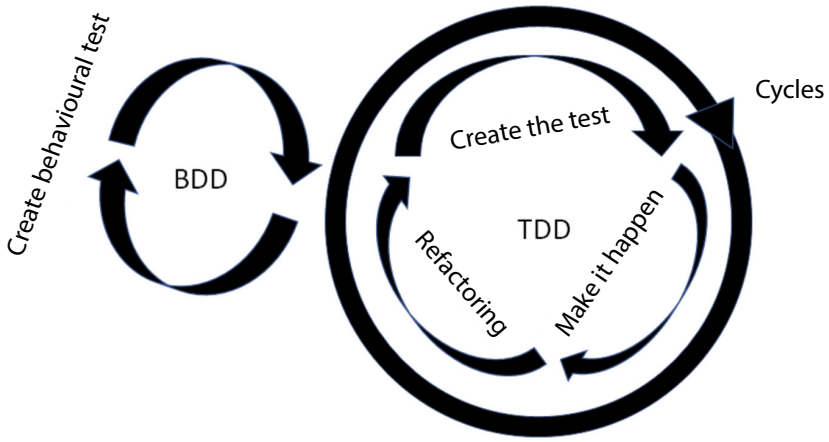

Figure 1. Behavior-directed development
Source: own elaboration.

3. Model

The model of the implemented cellular automata is briefly presented below. For more information on this, it is recommended to read the full definition in (Vargas-Forero *et al.*, 2019). The cellular automaton was defined as a tuple (G, S, N, R), that is, an ordered set of objects characterized by:

- (G): the grid
- (S): the states that a cell can take
- (N): the neighborhood
- (R): the transition rules

3.1. (G) Grid or graticule, where each cell can contain a pedestrian

- Each cell is a square and measures approximately 40 x 40, a value taken by suggestion in (Schadschneider, 2001).
- The number of pedestrians that can be in a cell is 0 or 1.
- The automaton is defined on a grid that represents the university campus and whose dimensions are 2589 rows by 1906 columns, which is obtained by dividing the dimension of the entire university campus by the size of the proposed cell.

3.2. (S) Finite set of states that a cell can take

An additional element considered in this definition of the model is the concept of impedance and it will be understood as the resistance (decrease in speed) that a pedestrian has when moving along a road or path, caused by reasons of a ramp, staircase or difficulty to cross the road.

For the proposed model, the ("") represent an empty cell equivalent to a non-passable path, the letter P represents a cell occupied by a pedestrian, the number 0 an empty cell of a passable path without impedance and when the cell has a value between 1 and n, this represents that the cell is empty, and the path is passable with impedance. The value of the impedance depends on the value of n, where 1 is the minimum impedance that delays the pedestrian one cycle for each movement, 2 would delay the pedestrian two cycles for each movement, and so on.

3.3. (N) Neighborhood corresponding to each cell

The project was based on the extended Moore neighborhood, which is an extension of the Von Newman neighborhood. This neighborhood contains cells on the diagonal as presented in Figure 2. To better understand the concept, see Wolfram (2019).

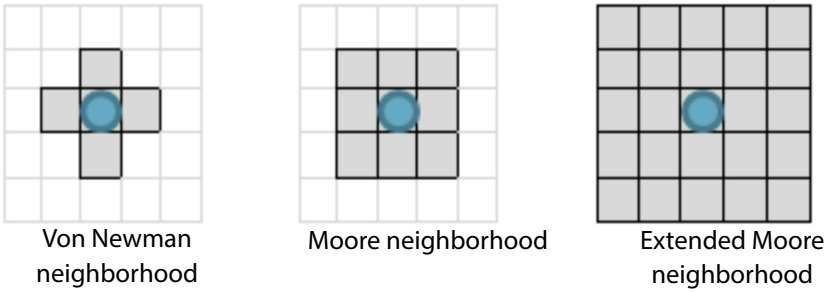

Figure 2. Neighborhood types
Source: own elaboration.

- Generate pedestrians.
- Pedestrian movement.
- Implementation of the rules of the cellular automaton model.
- Graphical interface that allows observing the behavior of pedestrians on the pedestrian network of the Melendez branch university campus

3.4. (R) Transition rules that define the cellular automaton model

The proposal of the model has the scope of the maximum neighborhood to 2 cells and not all the neighbors are considered within the possible movements. The pedestrian does not consider backing up on its path from origin to destination, so it only performs movements in one direction with two speeds. As seen in Figure 3, on the left side it is observed that the arrows only move one position and point only in the direction of origin and destination, and do not point from the destination to the origin; this indicates that pedestrians do not go back in the model and only move forward one position involving speed 1. On the right side you can see that it is similar to the one on the left, only that the movement is of two cells, which corresponds to speed 2.

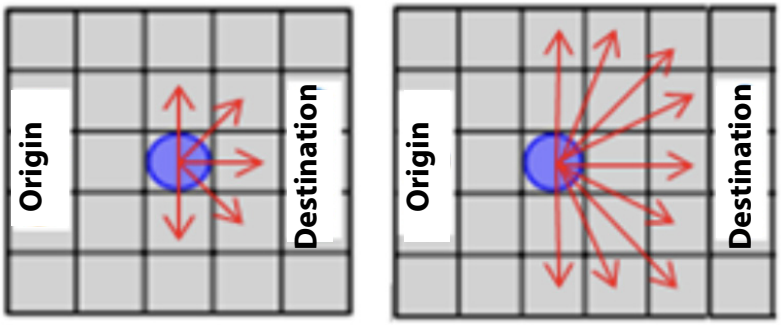

Figure 3. Speed types, left speed 1 and right speed 2
Source: own elaboration.

After the above, only a basic idea is presented to understand some elements of the implementation process. To understand this model in greater depth, it is recommended to see the article by Vargas-Forero *et al.* (2019), where they present how the pedestrian moves in the cellular automaton.

4. Results

Hereafter, some examples of the stages contemplated in the construction process of said development are presented, explained, and shown:

4.1. Planning

BDD is an agile development technique, oriented to the business and behavior of an application. Since its main objective is to ask what the application should do, inquiry techniques were used with the end users, in such a way that they allowed to formulate and write behavioral tests. To give clarity to the development, prototypes were made to provide a solution to one or more of the requirements. For the development to take place incrementally, it was decided to work from the most basic to the most complex requirements.

4.2. System requirements

To support the planning process, a preliminary specification of requirements was defined to establish the scope of the application. With the needs determined by the requirements, the scope of the application was delimited, and a clear approximation of the functionalities was estimated to fulfill the proposed objective. The solution must have the following functionalities:

- Represent the pedestrian network of the Meléndez branch university campus.
- Represent pedestrians.

4.3. Specification of the requirements

- The system must make it possible to represent the pedestrian network of the university campus. This network must include the 60 buildings and the pedestrian network (routes) corresponding to the cartography provided by the Research Group on Transportation, Traffic and Roads - GITTV.

- The system must allow to represent a pedestrian with the following characteristics: origin, destination, route, speed and current position in which the pedestrian is located.

- The system must allow generating pedestrians for each route, given an origin and destination, the initial and final time of generation of pedestrians, the number of pedestrians to generate, the probability of generation of pedestrians and the probability of generating speed pedestrians 1 and 2 as can be seen in Figure 3 of the model.

Pedestrian movement

- The system must provide a mechanism to simulate the movement of pedestrians.

- The system must provide a mechanism to resolve obstacles.
- The system must allow route segments to be configured to evaluate pedestrian flow.

- Given the coordinates of a segment of the route, the system must determine how many pedestrians pass in time T.

- The system must allow route segments to be configured to evaluate pedestrian density.

- Given the coordinates of a segment of the route, the system must determine how many pedestrians there are in a given time T.

- Given the location of a pedestrian on the grid that represents the Meléndez branch university campus, the system must determine if it is possible for a pedestrian to move, and which is its new location.

Verification of the rules of the cellular automaton model

- The system must allow calculating the distance of the pedestrian to his destination.

- Given the place of origin and destination of a pedestrian, the system must determine the orientation of the pedestrian.

- Given the current location of a pedestrian, the system must determine its next location and verify if that movement is outside of the established range.
- Given the current location of a pedestrian, the system must determine its next location and check if that movement is off the established path.

- Given the current location of a pedestrian, the system must determine its next location and verify if that movement is outside the established grid.

- Given a pedestrian who is in a current position and whose speed is 2, the system must evaluate if it can move and at what speed (speed 2, speed 1, not moving).

- Given a pedestrian who is in a current position and whose speed is 1, the system must evaluate if it can move (speed 1, not moving).
- The system must provide a graphical interface that allows observing the behavior of pedestrians on the pedestrian network of the Meléndez branch university campus.
- Once the execution of the system begins, it must show the grid that represents the Meléndez branch university campus and the pedestrian network.
- Given the coordinates of the upper left and the lower right corners, the system must show the scenario corresponding to said coordinates in the grid that represents the Meléndez branch university campus.
- The system must provide a pop-up menu, through which the user can select different previously established scenarios to observe the behavior of pedestrians.
- Show the movements of pedestrians.
- The system must verify the location of all pedestrians and update the interface, thus showing how pedestrians flow given an origin and destination within the pedestrian network that represents the Meléndez branch university campus.

4.4. Using BDD

The first thing that must be defined in BDD is the creation of the project scenarios, that is, the requirements of the project, where each of its needs is defined. The pedestrian out-of-range scenario is taken as an example, as it is a good example because it is easy to understand and explains the use of BDD. Table 1 shows some examples of the use of BDD for the cellular automaton:

Table 2.
BDD template. Check if the movement is outside the range of the matrix

Part	Description
Scenario Outline	Check if the move is outside the range of the matrix.
Given	At the beginning of the simulation, the rules of the cellular automaton are defined.
When	When you want to make a movement of a pedestrian, you must verify if its movement is within the defined matrix. As an example, for the matrix (<matriz>) where the pedestrian can move, the position <actual> of the pedestrian and the movement values in <movimientoxy>.
Then	It is determined that the pedestrian in the position <actual> with movement in <movimientoxy>, has a result (<resultado>) where 1 is a move out of range and 0 is not.

Source: own elaboration.

As it can be seen in Table 2, the scenario is presented in four parts which are explained below:

- **Scenario Outline:** a description of what this program functionality should do is presented. The case verifies if the movement is outside the range of the pedestrian matrix. That means that it should return true if the movement is outside the matrix.
- **Given**: The initial elements required for the functionality to be implemented are determined. For the scenario,an instance of the rules object is started. This rules object is created when starting the simulation.

- **When:** in this field, the moment and the elements necessary to carry out the functionality were defined. For this case, a matrix was defined where the pedestrian can move, the position where the pedestrian is and, finally, the position where t he pedestrian will be moved is needed.

- **Then:** finally, the expected result of the functionality was defined. For the example, the result with the given position of the pedestrian and the new position (x, y) of the same was used.

To complete the scenario, the tests that the implementation must pass are proposed.

As can be seen in Figure 4, all the examples to be carried out are presented to understand and test the functionality. Next, an explanation of each of them will be made. The first characteristic is the verification if a movement is within the possible range of a pedestrian.

Examples: outside of the matrix' range

matriz	actual	movimientoxy	result
0,0,0,1,1,0,1,1	0,0	0,0	0
0,0,0,1,1,0,1,1	0,0	1,0	0
0,0,0,1,1,0,1,1	1,1	1,0	1
0,0,0,1,1,0,1,1	0,0	2,2	1
0,0,0,1,1,0,1,1	0,0	1,1	0

Figure 4. Scenario 1 examples
Source: own elaboration.

Scenario: possible pedestrian movement.

- **Given**: the movement of a pedestrian from an origin position to a destination position and all possible movements.

- **When**: the target position is a possible move.

- **So**: the new position of the pedestrian is the target position.

Scenario: non-possible pedestrian movement.

- **Given**: the movement of a pedestrian from an origin position to a destination position and all possible movements.

- **When**: the target position is a possible move.

- **So**: the new position is the same current position.

So far, the first part of BDD ends, where the functionalities that need to be developed to solve the problem and complete the solution were defined. The next step was to build the necessary tests from the most elementary to the most complex to build the program.

Next, the test that was developed to give the solution of the problem is presented. The first part defines the use of the libraries that are needed in Python for the use of BDD, as well as the definition of the tag @given, in which the instance of the cellular automaton rules is started to define its movements.

```
from behave  import given, when, then

from hamcrest import assert_that, equal_to, is_not
```

```
from Reglas import Reglas

@given('the rules of the cellular automaton are defined')

def step_initia(context):

    context.objReglas = Reglas()
```

In this part of the test the tag @when is defined. In this label, the test data was collected, as well as the execution of the programs built to carry out the functionalities defined in the scenarios.

@When(´ to make a movement of a pedestrian, it must verify if its movement is within the defined matrix.

As an example, for the matrix ({matriz}) where the pedestrian is moving, the {actual} position of the pedestrian and the values of movement in ({movimientoxy}).')

```
def step_fueraDelRango(context, matriz, actual, movimientoxy):

    mat = matriz.split(',')

    sig = 0

    dic = {}

    tam = int(len(mat) /2)

    dic['tamano'] = (int(tam/2), int(tam/2) )


    for x in range (0, tam):

            dic[(int(mat[ sig ]), int(mat[ sig + 1 ]))] = 0

            sig + = 2

    x1 = int(actual.split(',') [0])

    y1 = int(actual.split(',') [1])

    x = int(movimientoxy.split(',')[0])

    y = int(movimientoxy.split(',')[1])

    context.resultadoMovimiento = context.objReglas.

    fueraDelRango (dic , [x1, y1], x, y )
```

As seen in the code shown above, most of it is to build the test with the data defined in the scenario, and only a single line of code was used to call the function that checks if the movement of the pedestrian is within the defined matrix for its movement.

Finally, the comparison of the test data was defined. In this code, it was observed that the result is only evaluated if the execution of the code is the same as expected in the previously defined scenario.

@then('it is determined that the pedestrian at position {actual}

With movement in ({movimientoxy}), has ({resultado})

where 1 is a movement out of range and 0 is not.')

def step_verifica_movimiento(context, actual, movimientoxy, resultado):

assert_that(bool(int(resultado)),

equal_to(context.resultMovimiento))

This test was applied for each of the defined data and the following console output is obtained as a result of the execution of the tests.

- **Scenario outline:** Check if the movement is outside the range of the matrix. -- @1.1 fuera_del_rango_ matriz # peatones.feature:31.
- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.
- **When:** When it is required to make a movement of a pedestrian, it must verify if its movement is within the defined matrix. As an example, for the array (0,0,0,1,1,0,1,1) where the pedestrian moves, the position 0,0 of the pedestrian and the movement values in (0,0). \ # steps \ step_reglas.py: 34.
- **Then:** it is determined that the pedestrian in position 0,0 with movement in (0,0), has (0) where 1 is a movement outside the range and 0 is not. # steps \ step \ _reglas.py: 50.
- **Scenario outline:** Check if the movement is outside the range of the matrix. -- @1.2 fuera_del_rango_matriz # peatones.feature:32.
- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.
- **When:** When it is required to make a movement of a pedestrian, it must verify if its movement is within the defined matrix. As an example, for the array (0,0,0,1,1,0,1,1) where the pedestrian moves, the position 0,0 of the pedestrian and the movement values in (0,0). \ # steps \ step_reglas.py: 34.
- **Then:** it is determined that the pedestrian in position 0,0 with movement in (1,0), has (0) where 1 is a movement outside the range and 0 is not. # steps \ step \ _reglas.py: 50.
-
- **Scenario Outline:** Check if the movement is outside the range of the matrix. -- @1.3 fuera_del_rango_ matriz # peatones.feature:33.
- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.
- **When:** When it is required to make a movement of a pedestrian, it must verify if its movement is within the defined matrix. As an example, for the array (0,0,0,1,1,0,1,1) where the pedestrian moves, the position 1,1 of the pedestrian and the movement values in (1,0). \ # steps \ step_reglas.py: 34.
- **Then:** it is determined that the pedestrian in position 1,1 with movement in (1,0), has (1) where 1 is a movement outside the range and 0 is not. # steps \ step \ _reglas.py: 50.

- **Scenario Outline:** Check if the movement is outside the range of the matrix. -- @1.4 fuera_del_rango_matriz # peatones.feature:34.
- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.
- **When:** When it is required to make a movement of a pedestrian, it must verify if its movement is within the defined matrix. As an example, for the array (0,0,0,1,1,0,1,1) where the pedestrian moves, the position 0,0 of the pedestrian and the movement values in (2,2). \ # steps \ step_reglas.py: 34.
- **Then:** it is determined that the pedestrian in position 0,0 with movement in (2,2), has (1) where 1 is a movement outside the range and 0 is not. # steps \ step \ _reglas.py: 50.
- **Scenario Outline:** Check if the movement is outside the range of the matrix. -- @1.5 fuera_del_rango_ matriz # peatones.feature:35.
- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.
- **When:** When it is required to make a movement of a pedestrian, it must verify if its movement is within the defined matrix. As an example, for the array (0,0,0,1,1,0,1,1) where the pedestrian moves, the position 0,0 of the pedestrian and the movement values in (1,1). \ # steps \ step_reglas.py: 34.
- **Then:** it is determined that the pedestrian in position 0,0 with movement in (1,1), has (0) where 1 is a movement outside the range and 0 is not. # steps \ step \ _reglas.py: 50.

In the exposed data, the execution of each of the proposed tests is observed, and it can be seen that all the tests passed correctly. To execute this test, the following code was implemented in Python:

```
import Peaton

class Reglas: # Transformation rules of the automaton.

# fueraDelRango: returns true if the values are outside the range of the array.

# param: matrix, pedestrian position, movement of "x" and "y" for the new position.

# return: true if the movement is invalid.

def fueraDelRango(self, matriz, position, x, y):

# Validate that the movement goes outside the range of the matrix

if ((posicion[0]+x> (matriz["tamano"] [0] -1))

or ((posicion[0] +x) <0)

or (posicion[1]+y> (matriz["size"] [1] -1))

or ((posicion[1] +y) <0)):

        return True return False
```

It could be seen that this code is quite simple and was selected for that purpose to simplify this part and explain its implementation of BDD. Next, two other scenarios are presented with their corresponding execution, which are relevant to the rules of the cellular automaton.

4.5. Execution of the scenario that verifies the progress of a pedestrian cell

The results of the execution of the test in BDD in Table 3 and the examples defined in Figure 5 are presented, for the scenario where the progress of a single pedestrian cell is verified.

Table 3.
BDD template. Advancement of a single pedestrian cell

Part	Description
Scenario Outline	A pedestrian moves one cell forward, if he can.
Given	The rules of the cellular automaton are defined.
When	When you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a pedestrian destination <destino> and the <actual> pedestrian position to advance one position.
Then	It is determined that the pedestrian in the <actual> position, has <resultado> of movement on the path for a step.

Source: own elaboration.

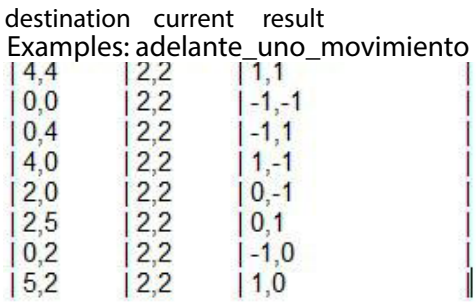

Figure 5. Examples of the advancement scenario of a single pedestrian cell
Source: own elaboration.

- **Scenario Outline:** A pedestrian advances one cell forward, if able. - @ 1.1 adelante_uno_movimiento # peatones.feature: 67.
- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.
- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 4,4 pedestrian destination and 2,2 pedestrian position to advance one position. # steps \ step_reglas.py: 76.
- **Then:** the pedestrian at position 2,2 is determined to have (1,1) movement on the road for one step. # steps \ step_reglas.py:89.
- **Scenario Outline:** a pedestrian advances one cell forward, if able. -- @1.2 adelante_uno_movimiento # peatones.feature:68.
- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.
- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 0,0 pedestrian destination and 2,2 pedestrian position to advance one position. # steps \ step_reglas.py: 76.

- **Then:** the pedestrian at position 2,2 is determined to have (-1,-1) movement on the road for one step. # steps \ step_reglas.py:89.

- **Scenario Outline:** a pedestrian advances one cell forward, if able. - @ 1.3 adelante_uno_movimiento # peatones.feature: 69.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 0,4 pedestrian destination and 2,2 pedestrian position to advance one position. # steps \ step_reglas.py: 76.

- **Then:** the pedestrian at position 2,2 is determined to have (-1,1) movement on the road for one step. # steps \ step_reglas.py:89.

- **Scenario Outline:** a pedestrian advances one cell forward, if able. - @ 1.4 adelante_uno_movimiento # peatones.feature: 70.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 4,0 pedestrian destination and 2,2 pedestrian position to advance one position. # steps \ step_reglas.py: 76.

- **Then:** the pedestrian at position 2,2 is determined to have (1,-1) movement on the road for one step. # steps \ step_reglas.py :89.

- **Scenario Outline:** a pedestrian advances one cell forward, if able. - @ 1.5 adelante_uno_movimiento # peatones.feature: 71.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 2,0 pedestrian destination and 2,2 pedestrian position to advance one position. # steps \ step_reglas.py: 76.

- **Then:** the pedestrian at position 2,2 is determined to have (0,-1) movement on the road for one step. # steps \ step_reglas.py :89.

- **Scenario Outline:** a pedestrian advances one cell forward, if able. - @ 1.6 adelante_uno_movimiento # peatones.feature: 72.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 2,5 pedestrian destination and 2,2 pedestrian position to advance one position. # steps \ step_reglas.py: 76.

- **Then:** the pedestrian at position 2,2 is determined to have (0,1) movement on the road for one step. # steps \ step_reglas.py:89.

- **Scenario Outline:** a pedestrian advances one cell forward, if able. - @ 1.7 adelante_uno_movimiento # peatones.feature: 73.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.
- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 0,2 pedestrian destination and 2,2 pedestrian position to advance one position. # steps \ step_reglas.py: 76.
- **Then:** the pedestrian at position 2,2 is determined to have (-1,0) movement on the road for one step. # steps \ step_reglas.py:89.
- **Scenario Outline:** a pedestrian advances one cell forward, if able. - @ 1.8 adelante_uno_movimiento # peatones.feature: 74.
- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.
- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 5,2 pedestrian destination and 2,2 pedestrian position to advance one position. # steps \ step_reglas.py: 76.
- **Then:** the pedestrian at position 2,2 is determined to have (1,0) movement on the road for one step. # steps \ step_reglas.py:89.

4.6. Execution of the scenario a pedestrian advances two cells forward

The following are the results of the BDD test run from Table 4 and the examples defined in Figure 6, for the scenario where the advance of two cells in front of the pedestrian is verified. Result of the execution of the test.

Table 4.
BDD template. A pedestrian moves two cells forward

Part	Description
Scenario Outline	A pedestrian moves two cells forward, if he can.
Given	The rules of the cellular automaton are defined.
When	When you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a pedestrian destination <destino> and the <actual> pedestrian position to advance two positions.
Then	It is determined that the pedestrian in the position <actual>, has<resultado> movement in movement on the road for two steps.

Source: own elaboration.

Examples: adelante_uno_movimiento

destination	current	result
4,4	2,2	2,2
0,0	2,2	-2,-2
0,4	2,2	-2,2
4,0	2,2	2,-2
2,0	2,2	0,-2
2,5	2,2	0,2
0,2	2,2	-2,0
5,2	2,2	2,0

Figure 6. A pedestrian moves two cells forward
Source: own elaboration.

- **Scenario Outline:** a pedestrian moves two cells forward, if able. - @ 1.1 adelante_dos_movimiento # peatones.feature: 83.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 4,4 pedestrian destination and 2,2 pedestrian position to advance one position. # steps \ step_reglas.py:95.

- **Then:** the pedestrian at position 2,2 is determined to have (2,2) movement on the road for two steps. # steps \ step_reglas.py:108.

- **Scenario Outline:** a pedestrian moves two cells forward, if able. - @ 1.2 adelante_dos_movimiento # peatones.feature: 84.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 0,0 pedestrian destination and 2,2 pedestrian position to advance two positions. # steps \ step_reglas.py:95.

- **Then:** the pedestrian at position 2,2 is determined to have (-2,-2) movement on the road for two steps. # steps \ step_reglas.py:108.

- **Scenario Outline:** a pedestrian moves two cells forward, if able. - @ 1.3 adelante_dos_movimiento # peatones.feature: 85.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When:** when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 0,4 pedestrian destination and 2,2 pedestrian position to advance two positions. # steps \ step_reglas.py:95.

- **Then:** the pedestrian at position 2,2 is determined to have (-2,2) movement on the road for two steps. # steps \ step_reglas.py:108.

- **Scenario Outline:** a pedestrian moves two cells forward, if able. - @ 1.4 adelante_dos_movimiento # peatones.feature: 86.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 4,0 pedestrian destination and 2,2 pedestrian position to advance two positions. # steps \ step_reglas.py:95.

- **Then:** the pedestrian at position 2,2 is determined to have (2,-2) movement on the road for two steps. # steps \ step_reglas.py:108.

- **Scenario Outline:** a pedestrian moves two cells forward, if able. - @ 1.5 adelante_dos_movimiento # peatones.feature: 87.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 2,0 pedestrian destination and 2,2 pedestrian position to advance two positions. # steps \ step_reglas.py:95.

- **Then:** the pedestrian at position 2,2 is determined to have (0,-2) movement on the road for two steps. # steps \ step_reglas.py:108.

- **Scenario Outline:** a pedestrian moves two cells forward, if able. - @ 1.6 adelante_dos_movimiento # peatones.feature: 88.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 2,5 pedestrian destination and 2,2 pedestrian position to advance two positions. # steps \ step_reglas.py:95.

- **Then:** the pedestrian at position 2,2 is determined to have (0,2) movement on the road for two steps. # steps \ step_reglas.py:108.

- **Scenario Outline:** a pedestrian moves two cells forward, if able. - @ 1.7 adelante_dos_movimiento # peatones.feature: 89.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 0,2 pedestrian destination and 2,2 pedestrian position to advance two positions. # steps \ step_reglas.py:95.

- **Then:** the pedestrian at position 2,2 is determined to have (-2,0) movement on the road for two steps. # steps \ step_reglas.py:108.

- **Scenario Outline:** a pedestrian moves two cells forward, if able. - @ 1.8 adelante_dos_movimiento # peatones.feature: 90.

- **Given:** the rules of the cellular automaton are defined # steps \ step_reglas.py: 10.

- **When**: when you want to make a movement of a pedestrian, you must verify if your movement is valid. As an example, given a 5,2 pedestrian destination and 2,2 pedestrian position to advance two positions. # steps \ step_reglas.py:95.

- **Then:** the pedestrian at position 2,2 is determined to have (2,0) movement on the road for two steps. # steps \ step_reglas.py:108.

4.7. Survey of the acceptance of the use of the methodology

As it was a first exercise in the use of BDD in this type of project, it was proposed to build a simple survey as a feedback instrument in the development process using BDD (see Table 5), for which the following results were obtained:

Table 5.
Methodology acceptance survey

Questions	Strongly disagree	Disagree	No, I'm sure	I agree	Strongly agree
Do you feel that the times proposed in the project planning were met?	0%	0%	0%	40%	60%
Did the model built in the project consider all the elements that you consider?	0%	0%	0%	20%	80%
Does the model work as you expect?	0%	0%	0%	30%	70%
Are you satisfied with the performance of the model building process?	0%	0%	0%	40%	60%
In your opinion, did you get feedback all the time on what was being done on the project?	0%	0%	0%	0%	100%
Could the prototype be adjusted during the project?	0%	0%	0%	40%	60%
Is the product quality as you expected?	0%	0%	0%	30%	70%
Would you recommend the use of the BDD methodology to other people?	0%	0%	0%	0%	100%
Would you use BDD again as a methodology for the development of a future project?	0%	0%	0%	0%	100%
Do you feel that there were elements that were missing in the construction of the model?	100%	0%	0%	0%	0%
Do you feel that the project was stalling and not advancing in the execution time?	80%	20%	0%	0%	0%

Source: own elaboration.

4.8. Execution of the model

As a final result, one of the executions of the cellular automaton is presented and only one of the executions is shown, because the campus of the Universidad del Valle is very large, this being the object of study for which the model was developed (Vargas-Forero *et al.*, 2019). It was decided to show the pedestrian path that leads to the central cafeteria, since it is in the rush hour (at noon) and it is the busiest in the entire university, showing the highest number of pedestrians and congestion on the road.

Figure 7 shows the pedestrian path of the central cafeteria, for this execution there are 1,121 pedestrians that come from many origins, but all converge to the destination of the central cafeteria. The execution time of this image is 5000 cycles, equivalent to 83.33 minutes, given that each cycle was assumed to be one second in time. It is observed that pedestrians flow smoothly and pedestrians flow in the same way in the rest of the paths.

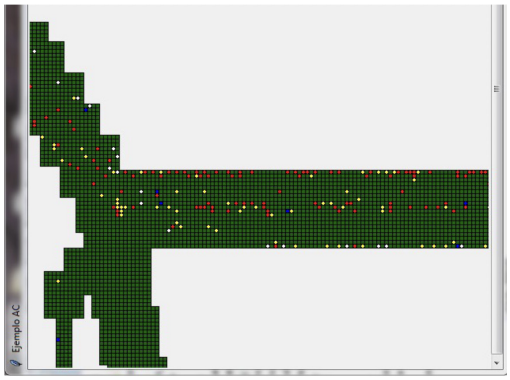

Figure 7. Execution of the cellular automaton with 1121 pedestrians and 5000 cycles
Source: own elaboration.

5. Conclusions

At the time of the project's development, the use of BDD in pedestrian movement simulation applications was not evidenced in the literature and less with the use of cellular automata; therefore, it was deemed appropriate to build the first cellular automata model for a simulation of pedestrian movement with the use of BDD.

As the first known implementation with the use of BDD had one of the highest expectations around it, the expected times or the requested characteristics were not met. However, at the end of the project, it was evident that these times were adequate and sufficient for the users who requested the software. The analysis and specification of the project requirements developed naturally for the people who participated in the project, considering that they had no training in computer science or systems. Thanks to continuous testing, product quality control could be carried out successfully, quickly, and efficiently.

In the construction stages, where the need to make changes due to new characteristics or modifications in the project was found, the implementation and integration in the development process was facilitated thanks to the fact that said modifications did not have an unfavorable impact on the product, there was no impact on the times or on the quality of the product. As it can be seen in the acceptance survey about the use of the methodology, the users who participated in the development showed a high acceptance in its use, since, for example, no question generated ambiguity or uncertainty for the user's response. Likewise, all responses were found in the ranges of greater acceptance (Agree and Strongly agree or Disagree and Strongly disagree), according to the orientation of the question.

The application of BDD in research projects could turn out to be wide and varied, especially with formal implementations. An example of future applications can be Event-B, where the transition from a requirements document to a formal specification can be made.

7. References

Beck, Kent; Beedle, Mike; Bennekum, Arie; Cockburn, Alistair; Cunningham, Ward; Fowler, Martin;… Thomas, Dave (2001). *Agilemanifesto. Manifiesto por el Desarrollo Ágil de Software.* Recuperado de https://agilemanifesto.org/iso/es/manifesto.html

Buitrago, Pablo; Kattán, José (2011). Universidad del Valle arquitectura para la educación. Cali, Colombia: Universidad del Valle.

Chen, Bi; Wang, Yafei; Wang, Donggen; Li, Qingquan; Lam, William; Shaw, Shih (2018). Understanding the impacts ofhuman mobility on accessibility usingmassivemobile phone tracking data. *Annals of the American Association of Geographers*, *108*(4), 1115–1133. https://doi.org/10.1080/24694452.2017.1411244

Engel, Jens; Rice, Benno; Jones, Richard (2012). *Behave.* Recuperado de https://behave.readthedocs.io/en/latest/

Ferguson-Smart, John (2014). BDD in Action: Behavior-Driven Development for the whole software lifecycle. Manning Publications.

Gipps, P. G.; Marksjö, B. (1985). A micro-simulation model for pedestrian flows. *Mathematics and Computers in Simulation*, *27*(2–3), 95–105. https://doi.org/10.1016/0378-4754(85)90027-8

Gudowski, Bartlomiej; Was, Jaroslaw (28-30 de junio de 2007). Some criteria of making decisions in pedestrian evacuation algorithms. En 6th International Conference on Computer Information Systems and Industrial Management Applications, CISIM 2007 (pp. 93–96), Minneapolis, MN, USA. https://doi.org/10.1109/CISIM.2007.61

Jurado, Carlos (2010). *Diseño Ágil con TDD*. Lulu.com.

Khamis, Abdelaziz; Abdelmonem, Ashraf (2002). The unified software development process and framework. *Dogus University Dergisi, 5,* 109-122. https://doi.org/10.31671/dogus.2019.348

Martins-Gonçalves, Natalia; Nuñez-Basantes, Alba (2017). Assessment the pedestrian accessibility in the brt stations in two cities of Latin America (breakout presentation). *Journal of Transport & Health*, *7*(Suplement), S76–S77. https://doi.org/10.1016/j.jth.2017.11.124

Muñoz-Ceballos, L. E. (2017). *Herramienta para modelar la accesibilidad peatonal al interior del campus de la Universidad del Valle con autómatas celulares sobre una plataforma SIG.* Universidad del Valle, Cali, Colombia.

North, D. (2006). Introducing {BDD}. https://dannorth.net/introducing-bdd/

Poppendieck, Mary; Cusumano, Michael (2012). Lean Software Development: A Tutorial. *IEEE Software*, *29*(5), 26–32. https://doi.org/10.1109/MS.2012.10766

Qin, Han; Curtin, Kevin; Rice, Matthew (2018). Pedestrian network repair with spatial optimization models and geocrowdsourced data. *GeoJournal*, *83*(2), 347–364. https://doi.org/10.1007/s10708-017-9775-x

Schadschneider, A. (2001). *Cellular Automaton Approach to Pedestrian Dynamics - Theory.* New York: Springer.

Solis, Carlos; Wang, Xiaofeng (2011). A Study of the Characteristics of Behaviour Driven Development. En *37th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 383–387), Oulu, Finland. https://doi.org/10.1109/SEAA.2011.76

Vargas-Forero,Victor; Muñoz-Ceballos, Luz; García-Baños, Ángel; Jaramillo-Molina, Ciro (2019). Modelo con autómatas celulares para analizar la accesibilidad peatonal al interior del campus universitario meléndez de la Universidad del Valle. *Scientia et Technica*, *24*(1), 67–75.

Was, J. (8-10 septiembre de 2005). Cellular automata model of pedestrian dynamics for normal and evacuation conditions. En *5th International Conference on Intelligent Systems Design and Applications 2005, ISDA '05* (154–159), Warsaw, Poland. https://doi.org/10.1109/ISDA.2005.31

Wolfram, M. (2019). *Moore Neighborhood*. http://mathworld.wolfram.com/MooreNeighborhood.html.