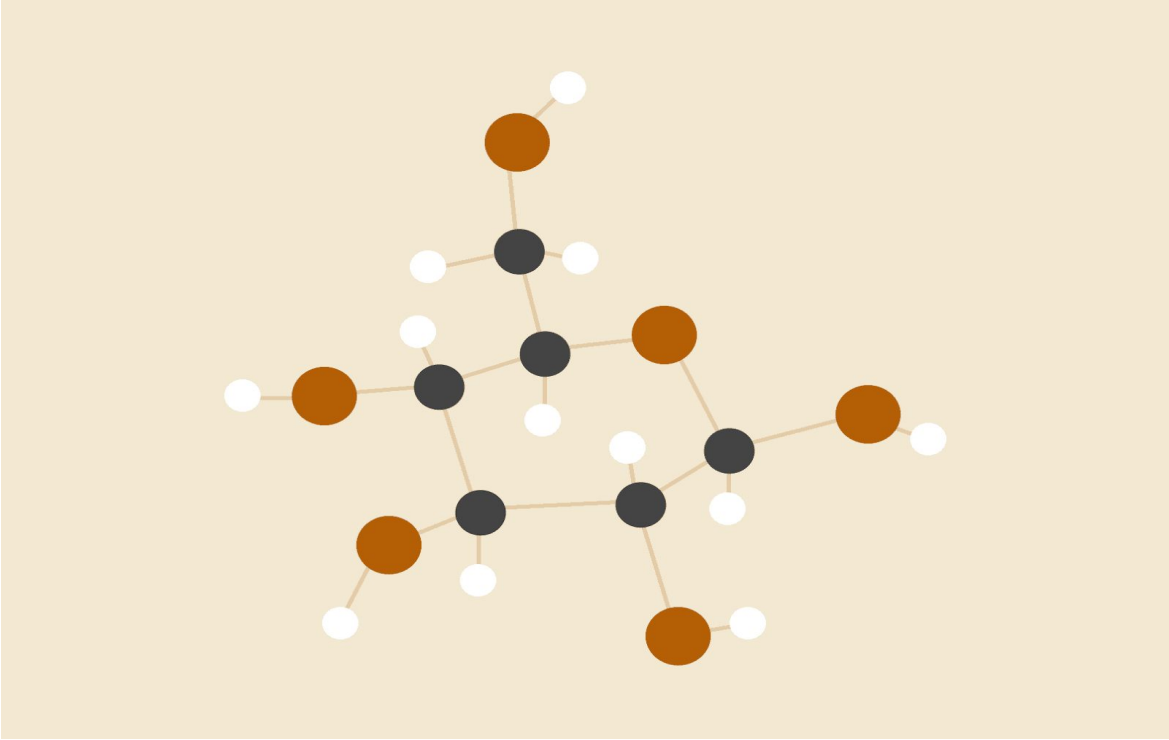
RELATÓRIO DE ATIVIDADE - The One

UFAM - Universidade Federal do Amazonas

ICC305 - Avaliação de Desempenho

Docente - Edjair de Souza Mota



Lucas Vinicius da Silva Reis

Moises Barbosa Gomes

Sthefanie Jofer Gomes Passo

06/2019

AGRADECIMENTOS

“Tudo posso naquele que me fortalece.” Filipenses 4:13

Em primeiro lugar agradecemos a Deus por ter nos dado sabedoria e harmonia para fazermos este trabalho, colocando pessoas que nos direcionaram para um melhor desempenho do trabalho, como o monitor Diogo Soares que nos indicou como utilizar o simulador e interpretar suas funções de forma correta. Nosso monitor precisou de muita paciência, mas sempre tinha prazer em nos ajudar e somos gratos por isso.

Algo que admitimos é que este trabalho demandou boa parte de nosso tempo durante duas semanas, mas graças a Deus, que renovou nossas forças, conseguimos realizá-lo e concluí-lo com excelência.

INTRODUÇÃO

A atividade teve como objetivo modelar um cenário de shopping com simulação de mobilidade de pessoas. Isso quer dizer que teríamos de criar o ambiente em uma linguagem/plataforma de desenvolvimento utilizando como auxílio as ferramentas PedSim (biblioteca de simulação de multidão de pedestres microscópica) e o The One (Simulador de ambiente de rede oportunista).

A especificação nos dá uma tabela de referência para que possamos nos basear e implementar:

Número de pessoas inicial	100
Taxa de renovação de pessoas	25 pessoas a cada 15 minutos
Número máximo de pessoas geradas	700
Tempo de simulação	6 horas
Velocidade das pessoas	0,3 ~ 1,5 m/s

HIPÓTESE

A princípio, teríamos que montar o mapa do shopping Manauara, desenhá-lo no PedSim, acrescentar os pedestres e a mobilidade dos mesmos. Em seguida, colocar no simulador The One. Porém o projeto ainda passou por algumas modificações antes de chegarmos ao resultado final.

MATERIAIS

1. Mapa do Manauara Shopping
2. Biblioteca do PedSim
3. Simulador The One
4. QT - Ferramenta de Desenvolvimento

Parte I - Modificar o PedSim

Para que conseguíssemos rodar o PedSim como queríamos, precisávamos modificar uma parte da biblioteca e testar conforme as mudanças iam se ajustando ao nosso trabalho.

Depois de pegar o mapa do Manauara, analisamos como poderia ser o cenário e adicionamos, parte por parte, ao PedSim.

Abaixo temos o código modificado da biblioteca:

```
// pedsim - A microscopic pedestrian simulation system.
// Copyright (c) by Christian Gloor


//g++ examples/trab2.cpp -o trab -lpedsim -L. -I. -std=c++11
//export LD_LIBRARY_PATH=.
//./trab

#include <iostream>
#include <cstdlib>
#include <chrono>
```

```
#include <thread>

#include "ped_includes.h"

#include "ped_outputwriter.h"

using namespace std;

int main(int argc, char *argv[]) {

    // create an output writer which will send output to a file
    Ped::OutputWriter *ow = new Ped::UDPOutputWriter();
    ow->setScenarioName("Manauara Shopping Simulator2");

    cout << "PedSim Exemplo utilizando a versao libpedsim " <<
Ped::LIBPEDSIM_VERSION << endl;

    // Setup CENA - Andar modelado sem obstáculo
    Ped::Tscene *pedscene = new Ped::Tscene(0, 0, 100, 30); // essa é a
cena (x1,y1) (x2,y2)
    pedscene->setOutputWriter(ow);
    const vector<Ped::Tagent*>& myagents = pedscene->getAllAgents();

    //Escrevendo dados no arquivo que vai pro the one
    cout << "360 360 0 100 0 30 0 0" << '\n';


    //PORTAS MANAUARA (X,Y,raio)
    Ped::Twaypoint *w2 = new Ped::Twaypoint(100, 20, 5);
    Ped::Twaypoint *w1 = new Ped::Twaypoint(0, 10, 5);

    // setup OBSTACULO
    pedscene->addObstacle(new Ped::Tobstacle(10, 10, 10, 20));//obstaculo 1
    pedscene->addObstacle(new Ped::Tobstacle(10, 10, 20, 10));
    pedscene->addObstacle(new Ped::Tobstacle(10, 20, 20, 10));

    pedscene->addObstacle(new Ped::Tobstacle(10, 30, 30, 15));//obstaculo 2
    pedscene->addObstacle(new Ped::Tobstacle(30, 15, 60, 30));
    pedscene->addObstacle(new Ped::Tobstacle(20, 0, 30, 10));
    pedscene->addObstacle(new Ped::Tobstacle(30, 10, 40, 0));

    pedscene->addObstacle(new Ped::Tobstacle(40, 10, 60, 10));//obstaculo 3
```

```
    pedscene->addObstacle(new Ped::Tobstacle(40, 10, 60, 20));
    pedscene->addObstacle(new Ped::Tobstacle(60, 10, 60, 20));

    pedscene->addObstacle(new Ped::Tobstacle(70, 10, 90, 10));//obstaculo 4
    pedscene->addObstacle(new Ped::Tobstacle(70, 10, 70, 20));
    pedscene->addObstacle(new Ped::Tobstacle(70, 20, 90, 10));

    pedscene->addObstacle(new Ped::Tobstacle(-10, 0, -10, 30));//obstaculo
5 contorno
    pedscene->addObstacle(new Ped::Tobstacle(110,30, -10, 30));
    pedscene->addObstacle(new Ped::Tobstacle(110,30, 110,  0));
    pedscene->addObstacle(new Ped::Tobstacle(-10, 0, 110,  0));


    //NUMERO DE PESSOAS INICIAIS = 100; máximo = 700
    //Ped::Tagent *a = new Ped::Tagent();
    for (int i = 0; i<100; i++) {//pessoas
        Ped::Tagent *a = new Ped::Tagent();
        //Onde o ponto vai surgir, na ENTRADA
        if (i%2==0) {
            a->addWaypoint(w1);
            double x = (double)(90.0 + rand()/(RAND_MAX/10.0));
            double y = (double)(10.0 + rand()/(RAND_MAX/10.0));
            a->setPosition( x, y, 0.0 );
            cout << 0 << ' ' << a->getid() << ' ' << x << ' ' << y <<'\n';
        }else {
            a->addWaypoint(w2);
            double x = (double)(0.0 + rand()/(RAND_MAX/10.0));
            double y = (double)(10.0 + rand()/(RAND_MAX/10.0));
            a->setPosition( x, y, 0.0 );
            cout << 0 << ' ' << a->getid() << ' ' << x << ' ' << y <<'\n';
        }

        a->setVmax(1.5);//velocidade
        pedscene->addAgent(a);

    }

    //cout << "Saiu do primeiro for" << '\n';

    // TEMPO 6h = 360min = 21600 segundos
    //Move all agents for 700 steps (and write their position through the
```

```
outputwriter)
    for (int i=1; i<=21600; i++) {
        //cout << "Entrou no segundo for."<< '\n';
        if (i%900==0) { //a cada 15min tem q renovar com mais 25 pessoas
                    //cout << "Entrou no if."<< '\n';
            for (int j = 0; j<25; j++) {//pessoas
                Ped::Tagent *a = new Ped::Tagent();
                //Onde o ponto vai surgir, na ENTRADA
                if (j%2==0) {
                    a->addWaypoint(w1);
                    double x = (double)(90.0 + rand()/(RAND_MAX/10.0));
                    double y = (double)(10.0 + rand()/(RAND_MAX/10.0));
                    a->setPosition( x, y, 0.0 );
                }else {
                    a->addWaypoint(w2);
                    double x = (double)(0.0 + rand()/(RAND_MAX/10.0));
                    double y = (double)(10.0 + rand()/(RAND_MAX/10.0));
                    a->setPosition( x, y, 0.0 );
                }

                a->setVmax(1.5);//velocidade
                pedscene->addAgent(a);

            }

        }

        for(Ped::Tagent *agente : pedscene->getAllAgents()){
          cout << i << ' ' << agente->getid() << ' ' <<
(agente->getPosition()).x << ' ' << (agente->getPosition()).y <<'\n';
        }

        pedscene->moveAgents(0.3 + rand()/(RAND_MAX/1.5) -
1.5);//velocidade de movimentação das pessoas
        std::this_thread::sleep_for(std::chrono::milliseconds(3));
        //for (auto a : myagents) if (a->reachedDestination()) delete
a;//se chegar na entrada, delete agente
    }

    // int notreached = myagents.size();
    // while (notreached > 0) {
    //     timestep++;
```

```
    //     notreached = myagents.size();
    //     pedscene->moveAgents(0.4);

    //     for (auto a : myagents) if (a->reachedDestination()) delete a;
    //     if (timestep >= 20000) notreached = 0; // seems to run forever.
    // }

    // Cleanup
    for (auto a : pedscene->getAllAgents()) { if (a->reachedDestination())
delete a; };
    //for (auto a : pedscene->getAllAgents()) { delete a; };
    for (auto o : pedscene->getAllObstacles()) { delete o; };
    delete pedscene;

    delete pedscene;
    delete w1;
    delete w2;
    delete ow;

    return EXIT_SUCCESS;
}
```

No código temos a adição dos obstáculos do mapa do Manaura, de acordo com o que está no Mapas da Google (ver Referência 1).

Primeiro as portas foram adicionadas, em seguida os contornos, no código comentados como “obstáculos 1 a 5”.

Por conseguinte, a adição de 100 pessoas. Com o tempo, a cada 15 minutos são adicionadas 25 pessoas, conforme solicitado na especificação.

Figura 1 - Exemplo no PedSim com a primeira versão da inserção dos dados na biblioteca da ferramenta

A Figura 1 mostra o resultado da modificação no PedSim.

Na primeira versão do código, obtivemos um erro. Ao criar um agente, não conseguimos deletar o agente quando passava pela porta.

Também havia erro de Segmentation Fault, por conta dos agentes criados não passarem de 573 pessoas, quando a especificação pedia 700 pessoas. Os agentes são ponteiros únicos, no código. Estávamos modificando o mesmo ponteiro que aponta pro

mesmo agente, inserindo múltiplos waypoints e setando sempre as mesmas posições no mesmo ponteiro e sempre re-adicionando ele na cena. Com ajuda do doutorando descobrimos o erro e ajustamos.

Segue o código modificado:

```
// pedsim - A microscopic pedestrian simulation system.
// Copyright (c) by Christian Gloor
//g++ examples/trab2.cpp -o trab -lpedsim -L. -I. -std=c++11
//export LD_LIBRARY_PATH=.
//./trab
#include <iostream>
#include <cstdlib>
#include <chrono>
#include <thread>

#include "ped_includes.h"

#include "ped_outputwriter.h"

using namespace std;

int main(int argc, char *argv[]) {

    // create an output writer which will send output to a file
    Ped::OutputWriter *ow = new Ped::UDPOutputWriter();
    ow->setScenarioName("Manauara Shopping Simulator2");

         cout << "PedSim Exemplo utilizando a versao libpedsim " <<
Ped::LIBPEDSIM_VERSION << endl;

    // Setup CENA - Andar modelado sem obstáculo
     Ped::Tscene *pedscene = new Ped::Tscene(0, 0, 100, 30); // essa é a cena
(x1,y1) (x2,y2)
    pedscene->setOutputWriter(ow);

    //Escrevendo dados no arquivo que vai pro the one
    cout << "360 360 0 100 0 30 0 0" << '\n';


    //PORTAS MANAUARA (X,Y,raio)
    Ped::Twaypoint *w2 = new Ped::Twaypoint(100, 20, 5);
    Ped::Twaypoint *w1 = new Ped::Twaypoint(0, 10, 5);

    // setup OBSTACULO
```

```
    pedscene->addObstacle(new Ped::Tobstacle(10, 10, 10, 20));//obstaculo 1
    pedscene->addObstacle(new Ped::Tobstacle(10, 10, 20, 10));
    pedscene->addObstacle(new Ped::Tobstacle(10, 20, 20, 10));

    pedscene->addObstacle(new Ped::Tobstacle(10, 30, 30, 15));//obstaculo 2
    pedscene->addObstacle(new Ped::Tobstacle(30, 15, 60, 30));
    pedscene->addObstacle(new Ped::Tobstacle(20, 0, 30, 10));
    pedscene->addObstacle(new Ped::Tobstacle(30, 10, 40, 0));

    pedscene->addObstacle(new Ped::Tobstacle(40, 10, 60, 10));//obstaculo 3
    pedscene->addObstacle(new Ped::Tobstacle(40, 10, 60, 20));
    pedscene->addObstacle(new Ped::Tobstacle(60, 10, 60, 20));

    pedscene->addObstacle(new Ped::Tobstacle(70, 10, 90, 10));//obstaculo 4
    pedscene->addObstacle(new Ped::Tobstacle(70, 10, 70, 20));
    pedscene->addObstacle(new Ped::Tobstacle(70, 20, 90, 10));

      pedscene->addObstacle(new  Ped::Tobstacle(-10,  0,  -10,  30));//obstaculo 5
contorno
    pedscene->addObstacle(new Ped::Tobstacle(110,30, -10, 30));
    pedscene->addObstacle(new Ped::Tobstacle(110,30, 110,  0));
    pedscene->addObstacle(new Ped::Tobstacle(-10, 0, 110,  0));


    //NUMERO DE PESSOAS INICIAIS = 100; máximo = 700
    //Ped::Tagent *a = new Ped::Tagent();
    for (int i = 0; i<100; i++) {//pessoas
        Ped::Tagent *a = new Ped::Tagent();
        //Onde o ponto vai surgir, na ENTRADA
        if (i%2==0) {
            a->addWaypoint(w1);
            double x = (double)(90.0 + rand()/(RAND_MAX/10.0));
            double y = (double)(10.0 + rand()/(RAND_MAX/10.0));
            a->setPosition( x, y, 0.0 );
            cout << 0 << ' ' << a->getid() << ' ' << x << ' ' << y <<'\n';
        }else {
            a->addWaypoint(w2);
            double x = (double)(0.0 + rand()/(RAND_MAX/10.0));
            double y = (double)(10.0 + rand()/(RAND_MAX/10.0));
            a->setPosition( x, y, 0.0 );
            cout << 0 << ' ' << a->getid() << ' ' << x << ' ' << y <<'\n';
        }

        a->setVmax(1.5);//velocidade
```

```
        pedscene->addAgent(a);

    }

    //cout << "Saiu do primeiro for" << '\n';

    // TEMPO 6h = 360min = 21600 segundos
      //Move all agents for 700 steps (and write their position through the
outputwriter)
    for (int i=1; i<=21600; i++) {
        //cout << "Entrou no segundo for."<< '\n';
        if (i%900==0) { //a cada 15min tem q renovar com mais 25 pessoas
                    //cout << "Entrou no if."<< '\n';
            for (int j = 0; j<25; j++) {//pessoas
                Ped::Tagent *a = new Ped::Tagent();
                //Onde o ponto vai surgir, na ENTRADA
                if (j%2==0) {
                    a->addWaypoint(w1);
                    double x = (double)(90.0 + rand()/(RAND_MAX/10.0));
                    double y = (double)(10.0 + rand()/(RAND_MAX/10.0));
                    a->setPosition( x, y, 0.0 );
                }else {
                    a->addWaypoint(w2);
                    double x = (double)(0.0 + rand()/(RAND_MAX/10.0));
                    double y = (double)(10.0 + rand()/(RAND_MAX/10.0));
                    a->setPosition( x, y, 0.0 );
                }

                a->setVmax(1.5);//velocidade
                pedscene->addAgent(a);

            }

        }

        for(Ped::Tagent *agente : pedscene->getAllAgents()){
           if(!agente->reachedDestination()) { //se o agente chegar no destino
não imprima ele (como se tivesse deletado)
                       cout << i << ' ' << agente->getid() << ' ' <<
(agente->getPosition()).x << ' ' << (agente->getPosition()).y <<'\n';
            }
        }
        pedscene->moveAgents(0.3 + rand()/(RAND_MAX/1.5) - 1.5);//velocidade de
movimentação das pessoas
```

```
        std::this_thread::sleep_for(std::chrono::milliseconds(3));
    }

    // int notreached = myagents.size();
    // while (notreached > 0) {
    //     timestep++;
    //     notreached = myagents.size();
    //     pedscene->moveAgents(0.4);

    //     for (auto a : myagents) if (a->reachedDestination()) delete a;
    //     if (timestep >= 20000) notreached = 0; // seems to run forever.
    // }

    // Cleanup
    for (auto a : pedscene->getAllAgents()) { delete a; };
    for (auto o : pedscene->getAllObstacles()) { delete o; };
    delete pedscene;
    delete w1;
    delete w2;
    delete ow;

    return EXIT_SUCCESS;
}
```

Após a modificação, conseguimos rodar o PedSim sem os erros e delimitamos todo o espaço do Manauara, a fim de garantir que o espaço fique totalmente visível para quem observa.

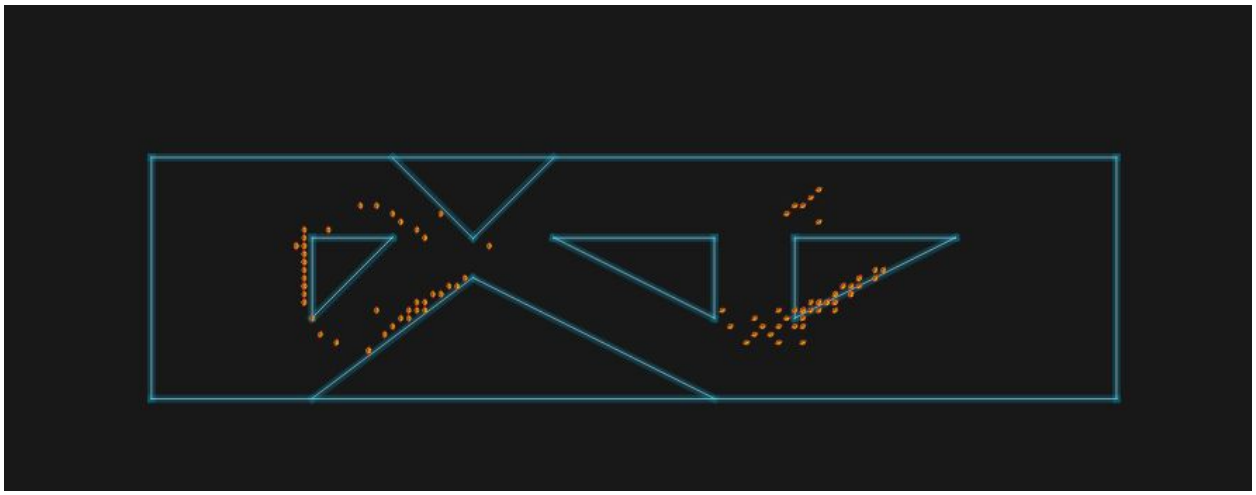


Figura 2 - Exemplo do PedSim após modificações

Na Figura 2, o 2° andar do Manauara Shopping é representado, já com as pessoas (ver como agentes) em movimento.

Mais exemplos:

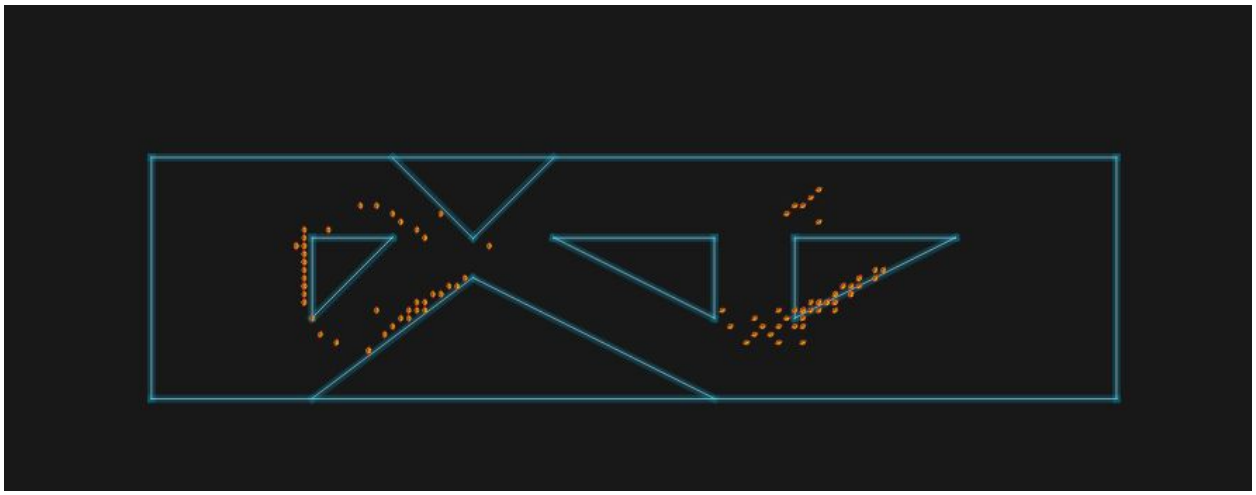


Figura 3 - Exemplo de simulação no PedSim pós-processamento

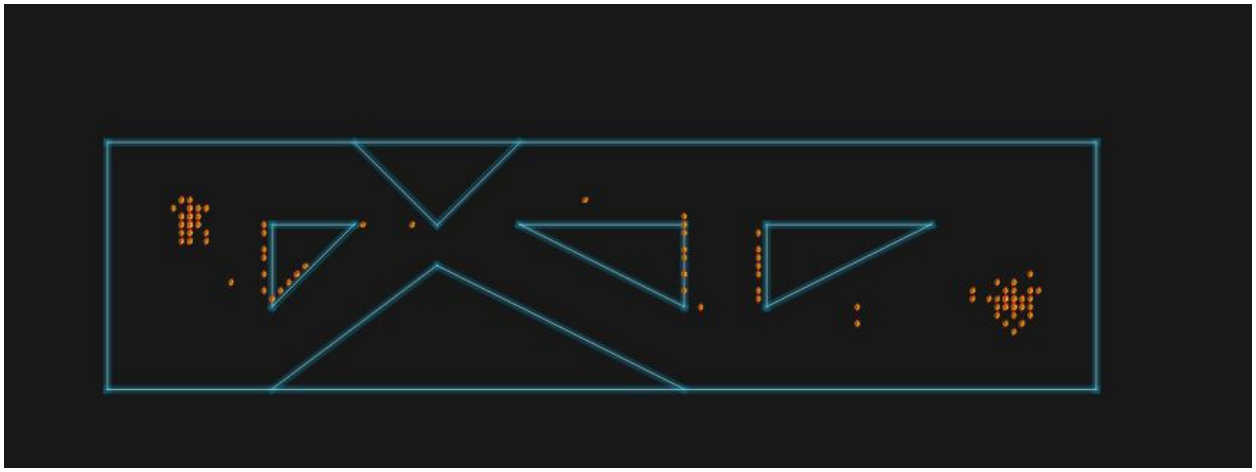


Figura 4 - Exemplo de simulação no PedSim pós-processamento

Parte II - Incluir a Modificação no Simulador TheOne

Para incluir as alterações no TheOne precisávamos do arquivo resposta do PedSim, e alterar alguns parâmetros do arquivo de configuração do próprio TheOne, com o intuito de que ele aceitasse e trabalhasse conforme precisávamos.

Abaixo o arquivo de configuração do TheOne com as configurações *default*.

```
#
# Default settings for the simulation
#

## Scenario settings
Scenario.name = default_scenario
Scenario.simulateConnections = true
Scenario.updateInterval = 1
# 43200s == 12h
Scenario.endTime = 43200

## Interface-specific settings:
# type : which interface class the interface belongs to
# For different types, the sub-parameters are interface-specific
# For SimpleBroadcastInterface, the parameters are:
# transmitSpeed : transmit speed of the interface (bytes per second)
# transmitRange : range of the interface (meters)

# "Bluetooth" interface for all nodes
btInterface.type = SimpleBroadcastInterface
# Transmit speed of 2 Mbps = 250kBps
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 1

# High speed, long range, interface for group 4
highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 10M
highspeedInterface.transmitRange = 1000

# Define 6 different node groups
Scenario.nrofHostGroups = 1

## Group-specific settings:
# groupID : Group's identifier. Used as the prefix of host names
# nrofHosts: number of hosts in the group
# movementModel: movement model of the hosts (valid class name from movement
package)
# waitTime: minimum and maximum wait times (seconds) after reaching destination
# speed: minimum and maximum speeds (m/s) when moving on a path
# bufferSize: size of the message buffer (bytes)
# router: router used to route messages (valid class name from routing package)
# activeTimes: Time intervals when the nodes in the group are active (start1,
end1, start2, end2, ...)
# msgTtl : TTL (minutes) of the messages created by this host group,
```

```
default=infinite

## Group and movement model specific settings
# pois: Points Of Interest indexes and probabilities (poiIndex1, poiProb1,
poiIndex2, poiProb2, ... )
#       for ShortestPathMapBasedMovement
# okMaps : which map nodes are OK for the group (map file indexes), default=all
#          for all MapBasedMovent models
# routeFile: route's file path - for MapRouteMovement
# routeType: route's type - for MapRouteMovement


# Common settings for all groups
Group.movementModel = ExternalMovement
#Passando o arquivo com os dados gerados pelo pedSim
ExternalMovement.file = dados_cena_manauara.txt

Group.router = EpidemicRouter
Group.bufferSize = 5M
Group.waitTime = 0, 120
# All nodes have the bluetooth interface
Group.nrofInterfaces = 1
Group.interface1 = btInterface
# Walking speeds
Group.speed = 0.5, 1.5
# Message TTL of 300 minutes (5 hours)
Group.msgTtl = 300

#Numero máximo de Pessoas
Group.nrofHosts = 700

# group1 (pedestrians) specific settings
Group1.groupID = p

# group2 specific settings
Group2.groupID = c
# cars can drive only on roads
Group2.okMaps = 1
# 10-50 km/h
Group2.speed = 2.7, 13.9

# another group of pedestrians
Group3.groupID = w
```

```
# The Tram groups
Group4.groupID = t
Group4.bufferSize = 50M
Group4.movementModel = MapRouteMovement
Group4.routeFile = data/tram3.wkt
Group4.routeType = 1
Group4.waitTime = 10, 30
Group4.speed = 7, 10
Group4.nrofHosts = 2
Group4.nrofInterfaces = 2
Group4.interface1 = btInterface
Group4.interface2 = highspeedInterface

Group5.groupID = t
Group5.bufferSize = 50M
Group5.movementModel = MapRouteMovement
Group5.routeFile = data/tram4.wkt
Group5.routeType = 2
Group5.waitTime = 10, 30
Group5.speed = 7, 10
Group5.nrofHosts = 2

Group6.groupID = t
Group6.bufferSize = 50M
Group6.movementModel = MapRouteMovement
Group6.routeFile = data/tram10.wkt
Group6.routeType = 2
Group6.waitTime = 10, 30
Group6.speed = 7, 10
Group6.nrofHosts = 2


## Message creation parameters
# How many event generators
Events.nrof = 2
# Class of the first event generator
Events1.class = MessageEventGenerator
# (following settings are specific for the MessageEventGenerator class)
# Creation interval in seconds (one new message every 25 to 35 seconds)
Events1.interval = 25,35
# Message sizes (500kB - 1MB)
Events1.size = 500M,1000M
# range of message source/destination addresses
Events1.hosts = 0,3
```

```
# Message ID prefix
Events1.prefix = M

#Events2.class = ExternalEventsQueue
#Events2.filePath = dados_3_nodes.txt


## Movement model settings
# seed for movement models' pseudo random number generator (default = 0)
MovementModel.rngSeed = 1
# World's size for Movement Models without implicit size (width, height;
meters)
MovementModel.worldSize = 4500, 3400
# How long time to move hosts in the world before real simulation
MovementModel.warmup = 1000

## Map based movement -movement model specific settings
MapBasedMovement.nrofMapFiles = 4

MapBasedMovement.mapFile1 = data/roads.wkt
MapBasedMovement.mapFile2 = data/main_roads.wkt
MapBasedMovement.mapFile3 = data/pedestrian_paths.wkt
MapBasedMovement.mapFile4 = data/shops.wkt

## Reports - all report names have to be valid report classes

# how many reports to load
Report.nrofReports = 2
# length of the warm up period (simulated seconds)
Report.warmup = 0
# default directory of reports (can be overridden per Report with output
setting)
Report.reportDir = reports/
# Report classes to load
Report.report1 = MessageStatsReport
Report.report2 = ContactTimesReport

## Default settings for some routers settings
ProphetRouter.secondsInTimeUnit = 30
SprayAndWaitRouter.nrofCopies = 6
SprayAndWaitRouter.binaryMode = true

## Optimization settings -- these affect the speed of the simulation
## see World class for details.
```

```
Optimization.cellSizeMult = 5
Optimization.randomizeUpdateOrder = true


## GUI settings

# GUI underlay image settings
GUI.UnderlayImage.fileName = data/helsinki_underlay.png
# Image offset in pixels (x, y)
GUI.UnderlayImage.offset = 64, 20
# Scaling factor for the image
GUI.UnderlayImage.scale = 4.75
# Image rotation (radians)
GUI.UnderlayImage.rotate = -0.015

# how many events to show in the log panel (default = 30)
GUI.EventLogPanel.nrofEvents = 100
# Regular Expression log filter (see Pattern-class from the Java API for
RE-matching details)
#GUI.EventLogPanel.REfilter = .*p[1-9]<->p[1-9]$
```

O arquivo de configuração do The One que contém os dados de entrada segue o formato abaixo:

```
/**
 * Reader for ExternalMovement movement model's time-location tuples.
 * <P>
 * First line of the file should be the offset header. Syntax of the header
 * should be:<BR>
 * <CODE>minTime maxTime minX maxX minY maxY minZ maxZ</CODE>
 * <BR>
 * Last two values (Z-axis) are ignored at the moment but can be present
 * in the file.
 * <P>
 * Following lines' syntax should be:<BR>
 * <CODE>time id xPos yPos</CODE><BR>
 * where <CODE>time</CODE> is the time when a node with <CODE>id</CODE>
should
 * be at location <CODE>(xPos, yPos)</CODE>.
 * </P>
 * <P>
 * All lines must be sorted by time. Sampling interval (time difference
between
```

```
* two time instances) must be same for the whole file.
* </P>
*/
```

O arquivo gerado segue esse padrão para todos os 700 agentes (pessoas) da ferramenta.

Parte III - Rodar o Simulador TheOne

Após todos os dados de entrada serem inseridos no TheOne, obtivemos a simulação, com a duração de 6 horas, conforme pedido.
Seguem algumas imagens da simulação do TheOne:

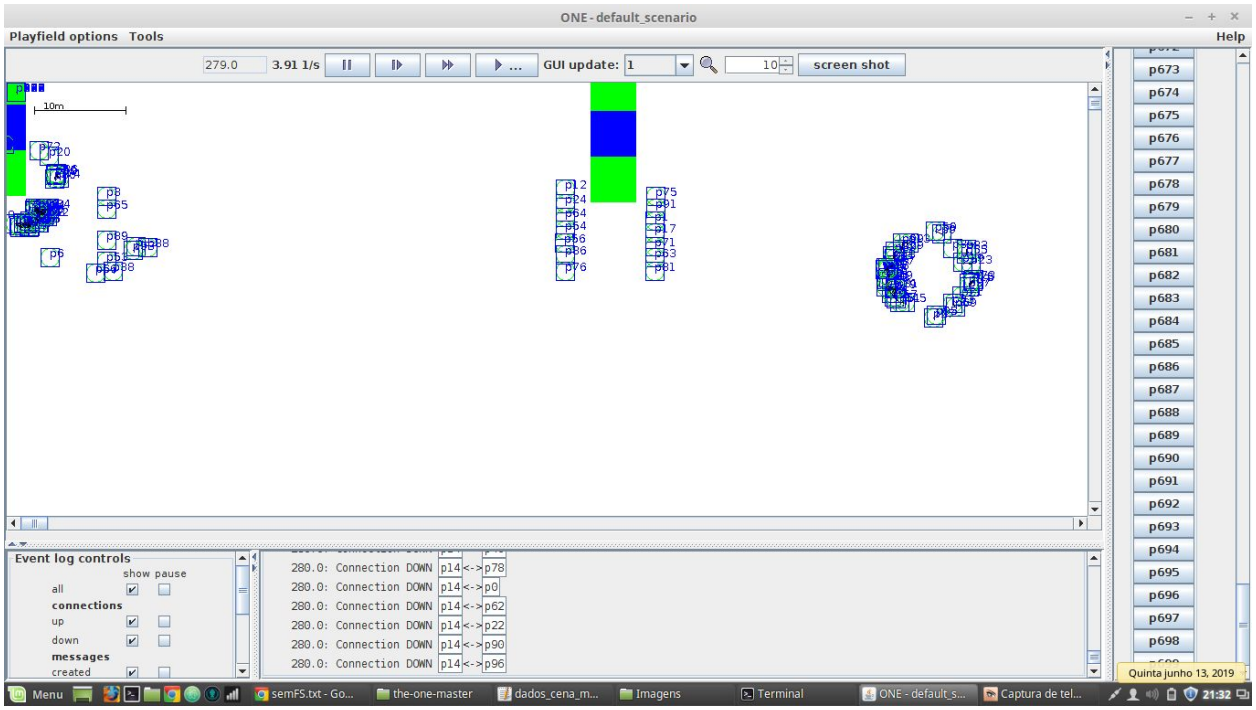


Figura 5 - Print de tela, simulação do TheOne

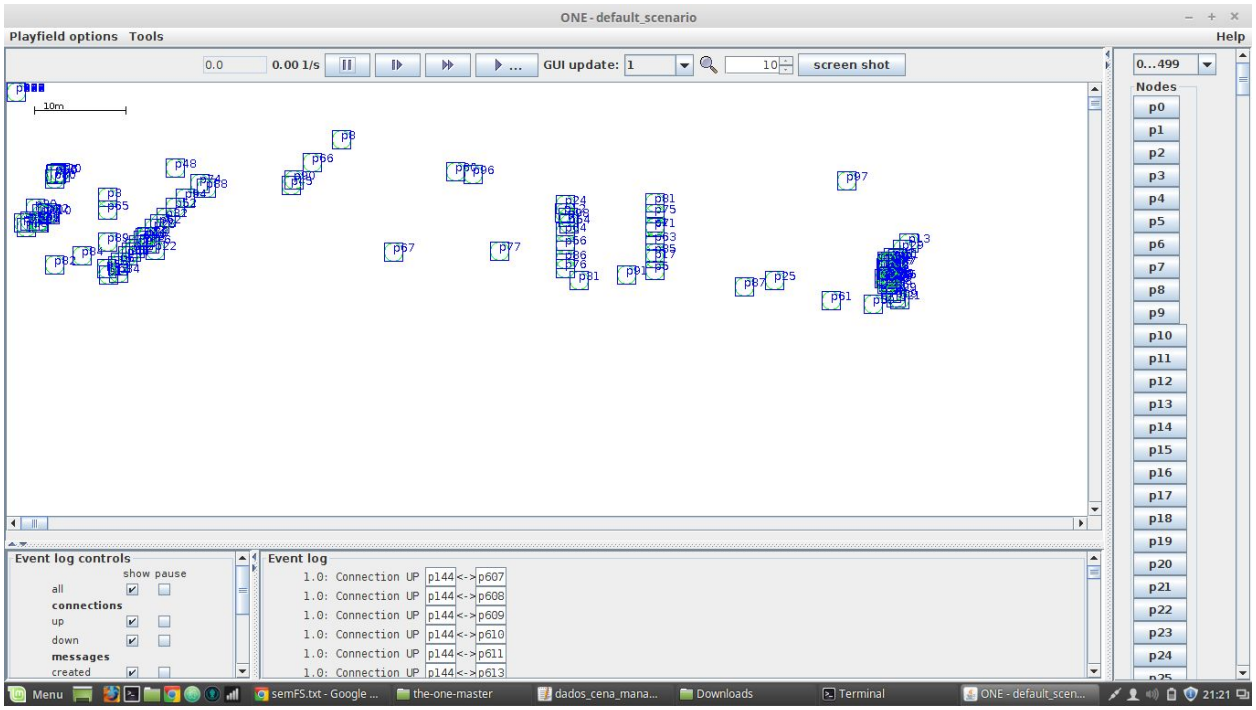


Figura 7- Print de tela, exemplo de simulação

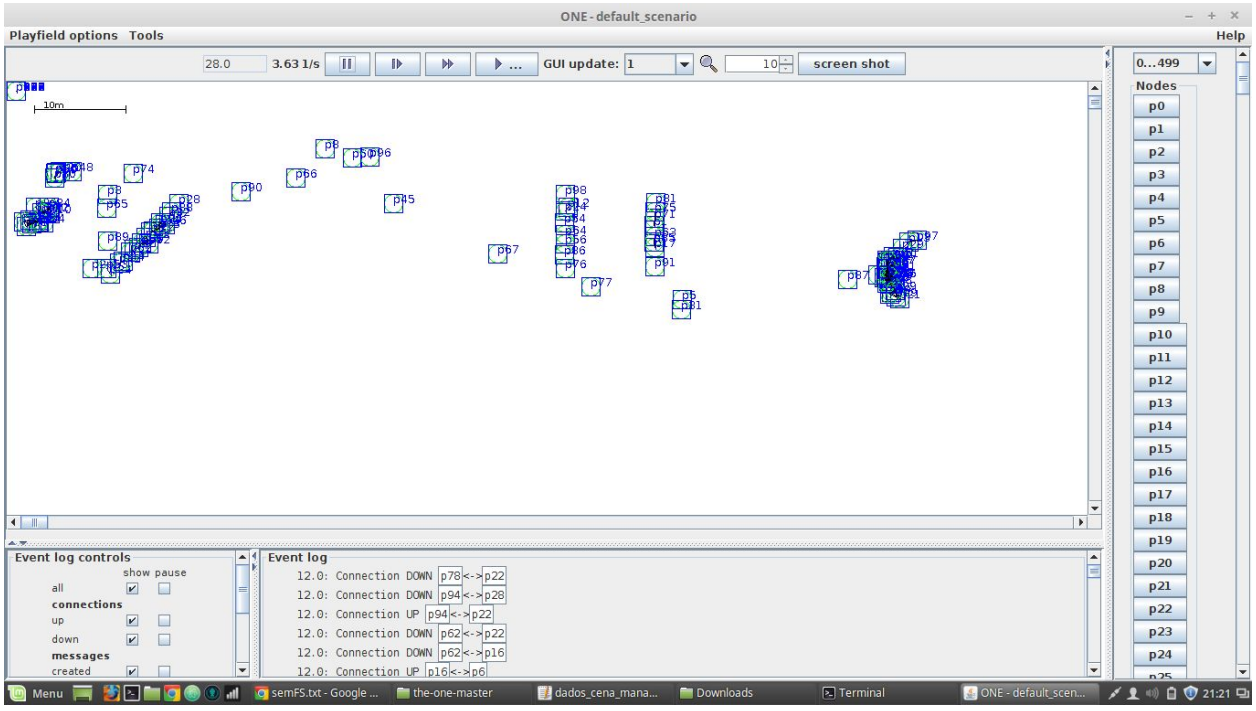


Figura 8- Simulação do TheOne

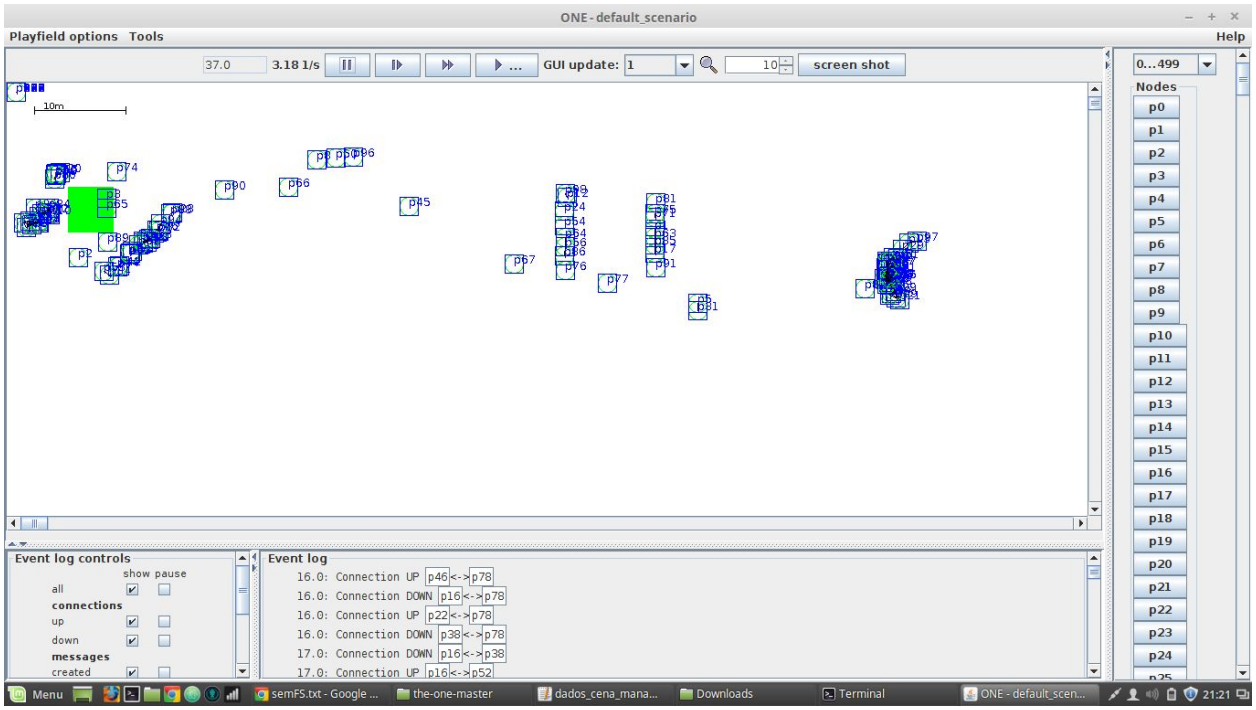


Figura 9 - Simulação do TheOne

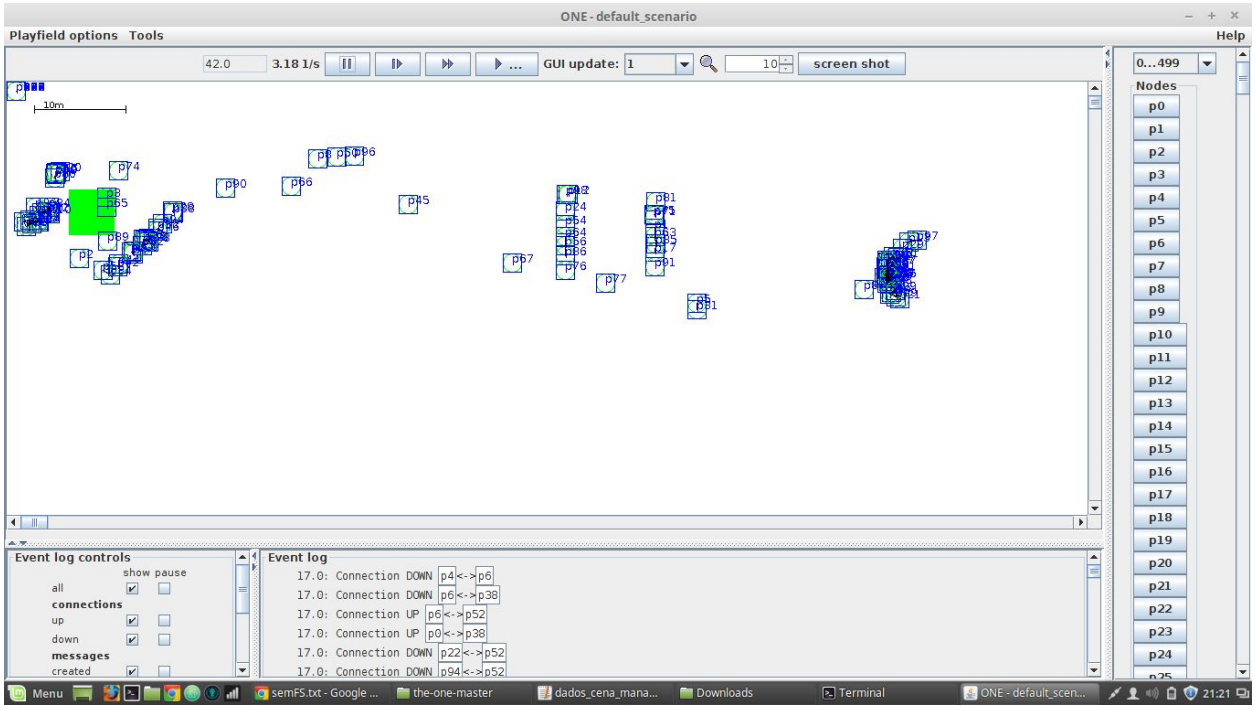


Figura 10 -Simulação do TheOne

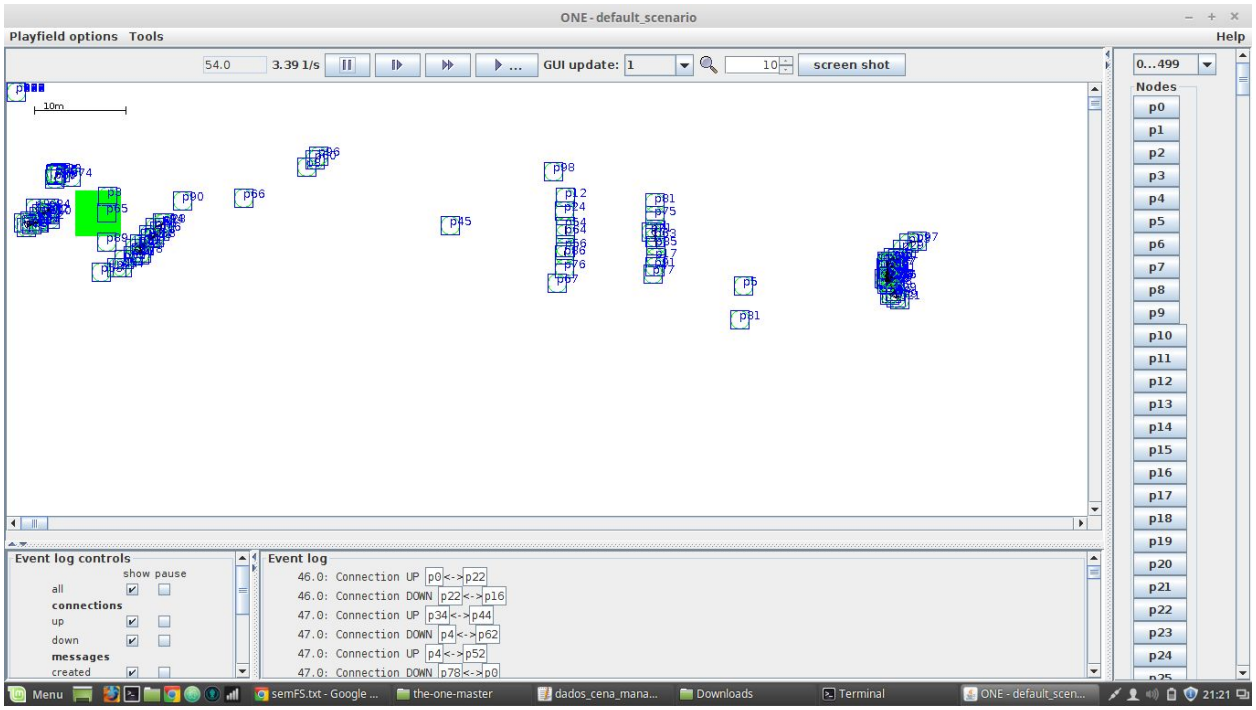


Figura 11 - Simulação do TheOne

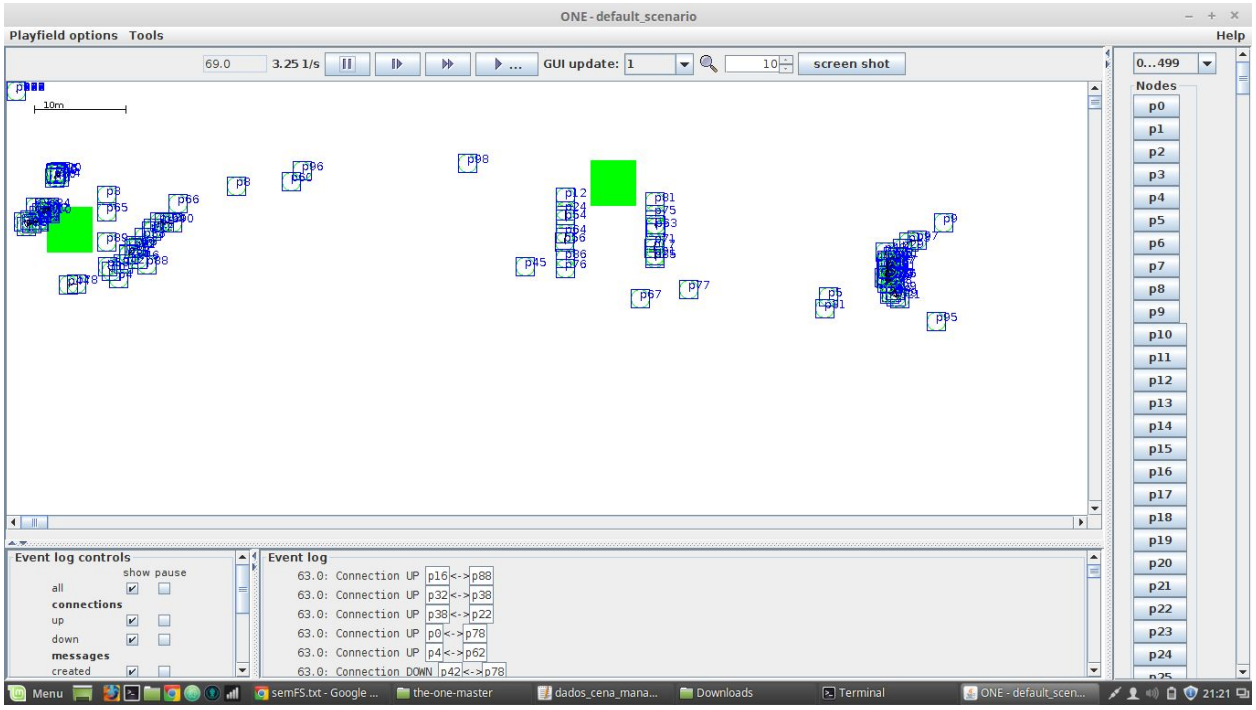


Figura 12 - Simulação do TheOne

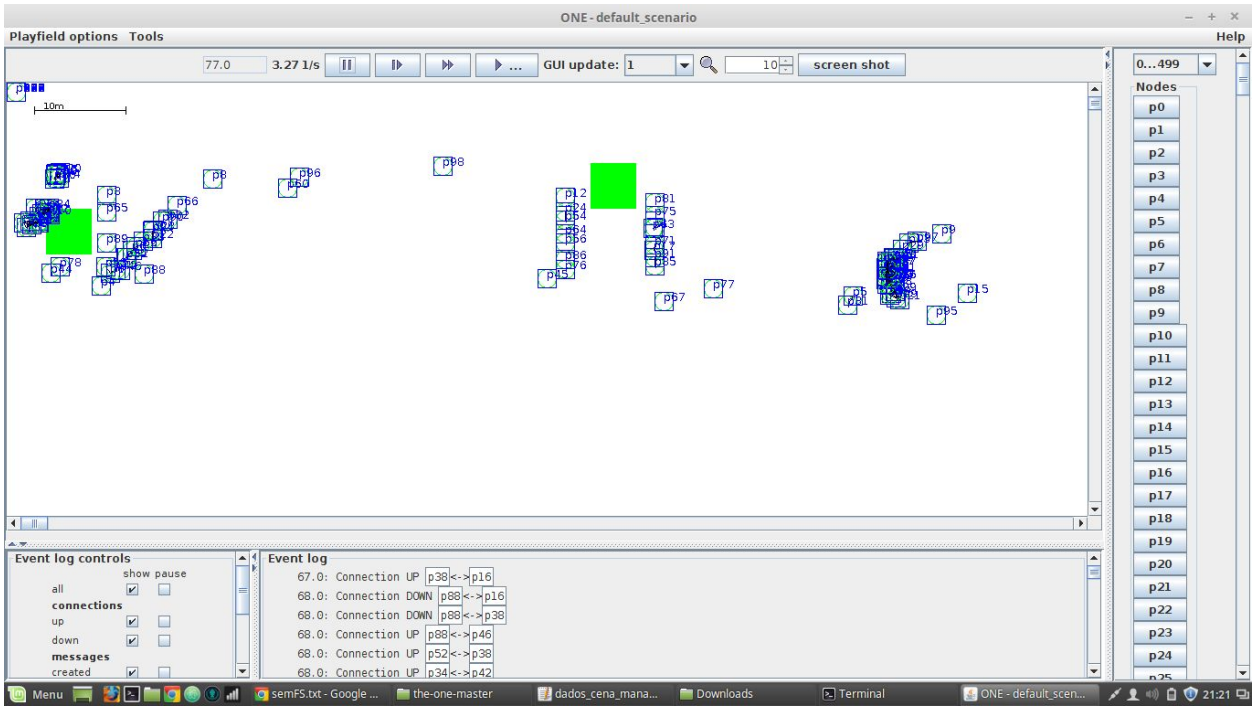


Figura 13 - Simulação do TheOne-

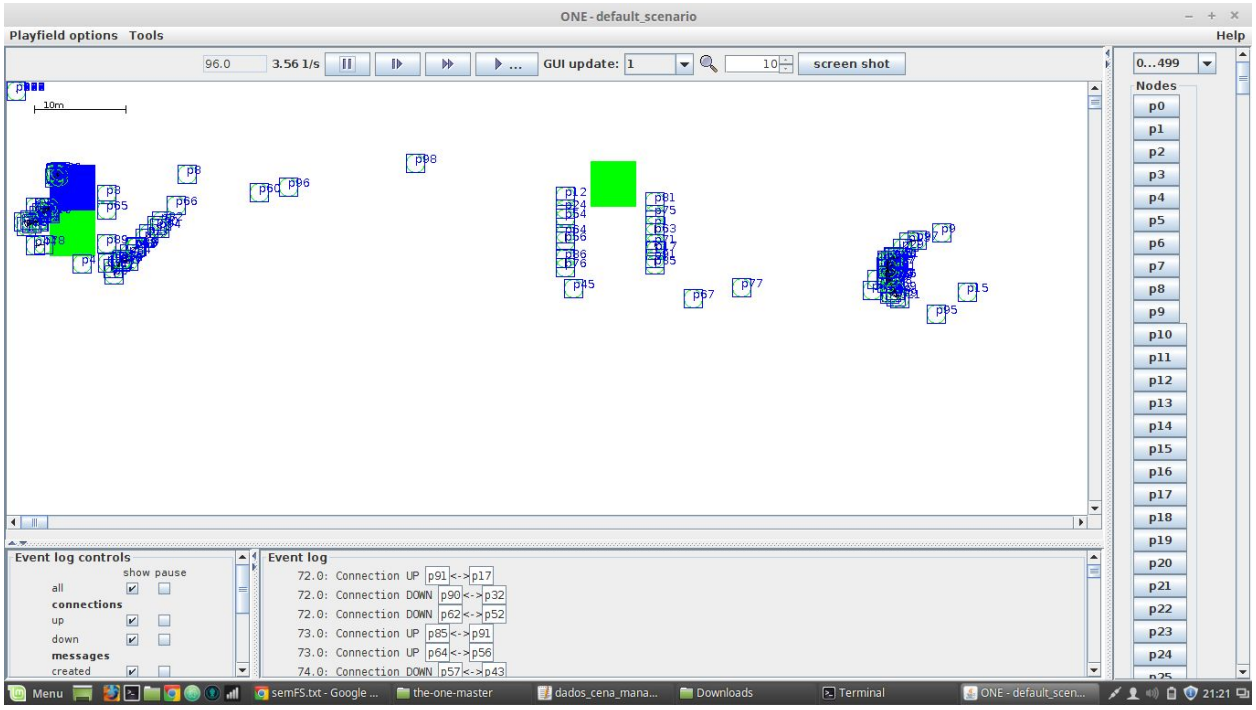


Figura 14 - Simulação do TheOne

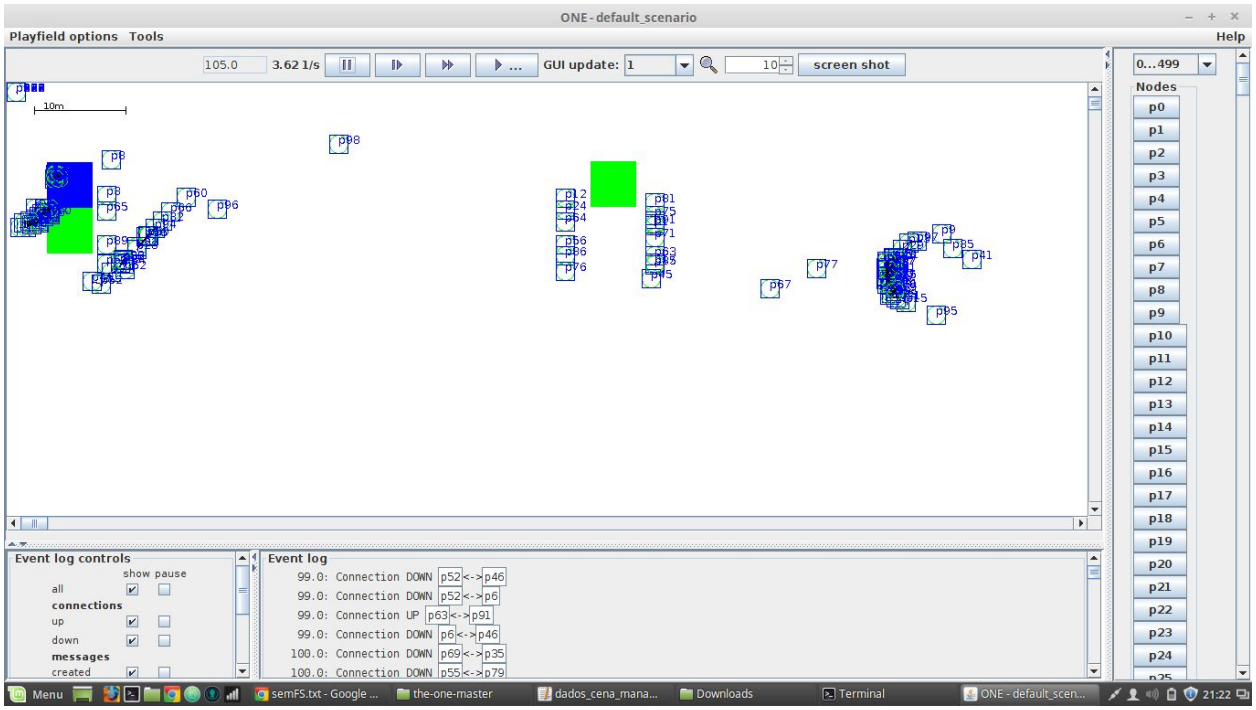


Figura 15 - Simulação do TheOne

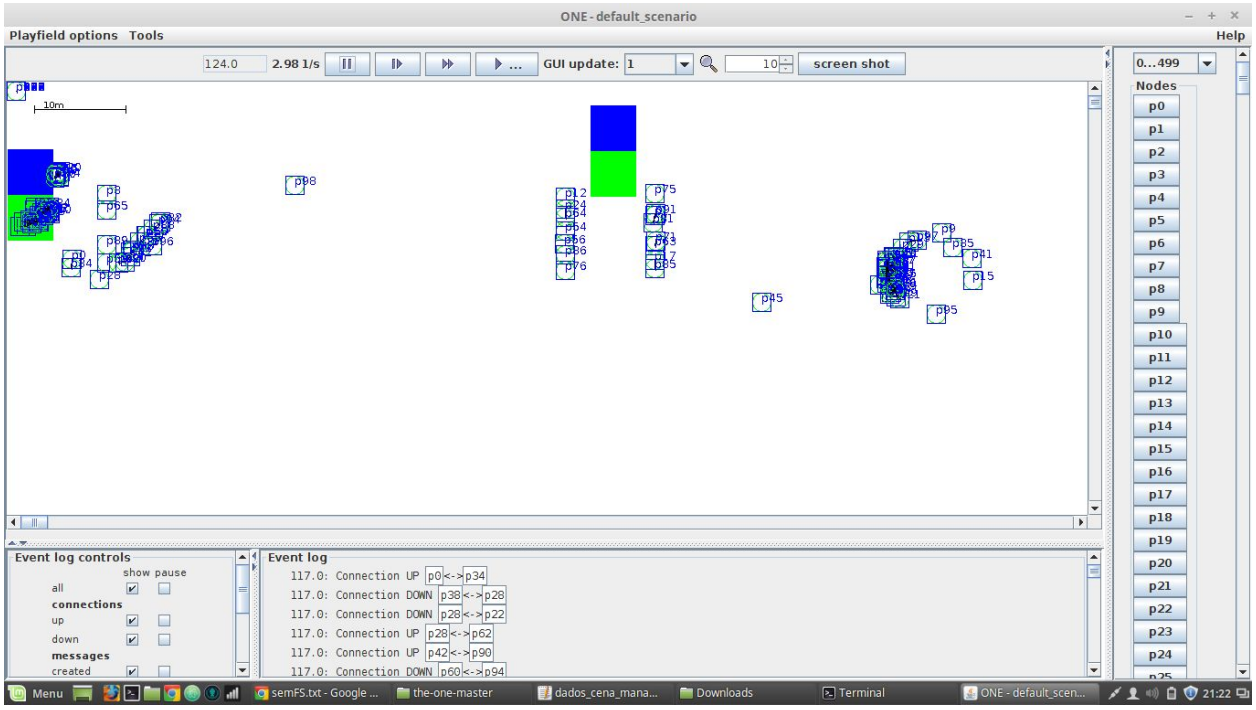


Figura 16 - Simulação do TheOne

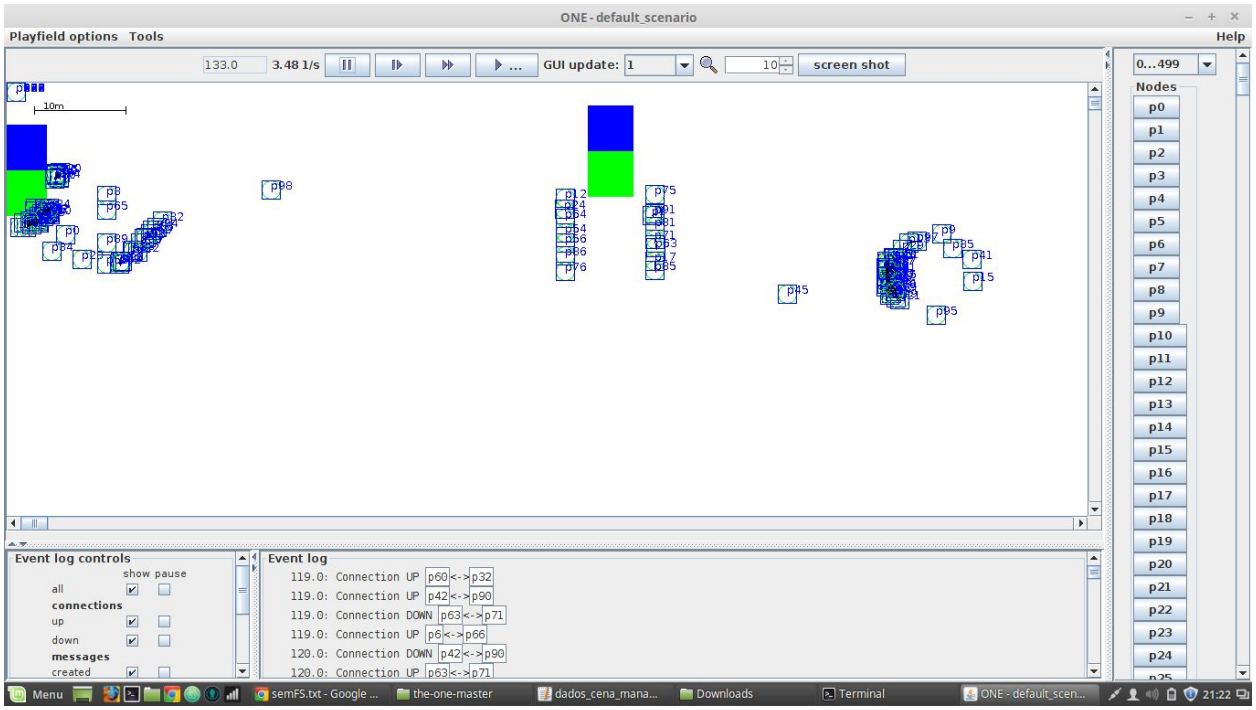


Figura 17 - Simulação do TheOne

CONCLUSÃO

O uso de simulações atualmente é muito usado para mensurar o desempenho em sistemas do mundo real que utilizam grandes recursos para sua execução. O método de simular o funcionamento de sistemas de grandes eventos é utilizado como uma alternativa para redução de gastos envolvidos em uma simulação real.

Diante destes aspectos nosso trabalho foi realizar a simulação da circulação de um grande número de pessoas em um shopping Center da cidade de Manaus, o Manauara Shopping.

Por ser um grande centro comercial e de entretenimento, o shopping recebe grande fluxo de movimentação de pessoas, principalmente em horários próximos ao entardecer e durante a noite. Porém para nosso sistema resolvemos normalizar esse fluxo. Adotamos um fluxo constante de acordo com o que foi solicitado na especificação do trabalho (25 novas pessoas a cada 15 minutos). Considerando este aspecto, a simulação não se aproximou da realidade tanto quanto poderia, porém para efeitos de simulação e mensuração do fluxo de clientes no shopping a simulação atendeu ao

objetivo.

Alguns problemas foram encontrados durante a execução dos scripts. Um deles foi em relação a localização de pessoas dentro e fora do estabelecimento, pois ao setar a localização das pessoas fora do shopping o simulador ainda as considerava como participantes da simulação. O problema foi resolvido utilizando uma função que detecta a aproximação do nó (pessoa) nas entradas e saída do shopping, ou seja, se a pessoa estivesse no raio do ponto de entrada ou de saída o simulador considerava que a pessoa estava fora da área delimitada para a simulação.

Conseguimos reter o aprendizado que esta atividade propõe de simular um problema do mundo real em plataformas virtuais. Fomos capazes de identificar o problema e o modo como resolvê-lo, abstraindo informações dos erros gerados pelas simulações-testes para ter melhor resultado nas próximas execuções.

REFERÊNCIAS

1. https://www.google.com/maps/search/manauara+shopping/@-3.104308 1,-60.0124342,18.25z - Mapa do Manauara, vista de satélite
2. http://pedsim.silmaril.org/
3. www.qt.io