Akademia Górniczo-Hutnicza im.
Stanisława Staszica w Krakowie

Social force model for pedestrian dynamics

Marcin Jakubowski
Minh Nhat Trinh

Prowadzący
Dr hab. inż. *Jarosław Wąs*

16 stycznia 2018

Spis treści

1 Wprowadzenie do tematu **2**

2 Proponowany model zjawiska **3**
2.1 Koncepcja 'Social force' i cele modelu 3
2.2 Formułowanie modelu 'Social force' 4
2.2.1 Ogólne równanie 4
2.2.2 Składowe siły 'Social Force' 4
2.3 Diagramy 6
2.3.1 Diagramy cykli działań 6
2.3.2 Diagramy przypadków użycia 7

3 Symulacja komputerowa **8**
3.1 Język i narzędzia programowania 8
3.2 Implementacja 9
3.2.1 Implementacja modelu 9
3.2.2 Interfejs użytkownika 10
3.2.3 Różnice implementacji w porównaniu z wersją książkową 11
3.2.4 Inicjalizowanie sceny 12
3.3 Diagramy klas UML 14

4 Wyniki symulacji **15**
4.1 Wyniki i statystyki 15
4.2 Zastosowane procedury kalibracji i walidacji 17

5 Wnioski **18**
5.1 Zebranie najważniejszych wniosków 18
5.2 Zebranie najważniejszych wyzwań i trudności rozpatrywanego problemu 18
5.3 Future Works 19

6 Podziękowanie **20**

Rozdział 1

Wprowadzenie do tematu

W ciągu ostatnich dwóch dekad modele opisujące ruch pieszych znalazły znaczące zainteresowani. Mają one bowiem ogromne znaczenie podczas projektowania i planowania stref masowego użytku publicznego, np. metra lub stacji kolejowych, stadionów, centr handlowych, dzięki możliwości zasymulowania ewakuacji czy innych zjawisk jak problem wąskiego gardła, rozładowania tłumu.

Wszystkie wielkości modelowe, takie jak współrzędne $\vec{r}$ i prędkość $\vec{v}$ pieszych, są możliwe do zmierzenia, a zatem dane symulacyjne są porównywalne z danymi eksperymentalnymi, co daje łatwość w kalibracji modelu symulacyjnego.

W przeszłości wiele osób podejmowało projekty badawcze związane z opracowaniem modelu najlepiej opisującego ruch pieszych np. *Couzin & Krause 2003; Ball 2004; Sumpter 2006; Helbing & Molnar 1995, 2008; itd.*.

W następnym paragrafie przedstawimy model Ścial Force"dla ruchu pieszych zaproponowany pierwotnie w 1995 roku przez Helbinga i Molnara. Model ten opiera się na koncepcji Ścial Force"jako siły oddziałującej na pieszego pochodzącej od interakcji z innymi pieszymi w tłumie czy interakcji otoczeniem np. ścianami.

W jaki sposób piesi modyfikują swoje zachowanie w odpowiedzi na interakcje z innymi? Odpowiedź na to pytanie pozwala zrozumieć mechanizmy prowadzące do samoorganizacji w tłumie i pomaga budować niezawodne modele.

Rozdział 2

Proponowany model zjawiska

2.1 Koncepcja 'Social force' i cele modelu

Wiele osób ma poczucie, że ludzkie zachowania są *chaotyczne* lub przynajmniej bardzo nieregularne i nieprzewidywalne. W rzeczywistości, szczególnie w dużych zbiorowiskach, ludzkie zachowanie można opisać jako model matematyczny, a w związku z tym da się przewidzieć zachowanie ludzi.

Sugeruje się, że ruch pieszych można opisać tak, jak gdyby podlegał 'Social Force'. Te siły nie są bezpośrednio wywierane przez środowisko zewnętrze pieszego, ale odzwierciedlają one wewnętrzne motywacje pieszego do wykonywania określonych czynności jako odpowiedź na interakcje ze środowiskiem zewnętrznym. W prezentowanym modelu zachowań pieszych zasadnicze znaczenie ma kilka sił. Po pierwsze, konstrukcja opisująca siłę związaną z oczywistym dążeniem pieszego do wyjścia czy przejścia. Po drugie, konstrukcja odzwierciedlająca wpływ innych pieszych, między innymi związana z zachowaniem pewnej odległości pomiędzy pieszymi, tzw. strefy komfortu, a także umożliwiającej pieszemu wykonanie kolejnego kroku. Po trzecie, konstrukcja będąca odzwierciedleniem wpływu wszelkiego rodzaju przeszkód np. ścian. Często dodaje się również składową odzwierciedlającą siłę przykuwania uwagi pieszego do czegoś interesującego, a także siłę odzwierciedlającą to, że często piesi poruszają się w grupach znajomych.

Komputerowe symulacje ruchu oddziałujących na siebie pieszych pokazują, że model 'Social force' jest bardzo realistycznie.

2.2 Formułowanie modelu 'Social force'

2.2.1 Ogólne równanie

Zgodnie z koncepcją 'Social force' przez *Helbing & Molnar 1995* możemy uznać, że ruch pieszego można opisać za pomocą trzech różnych składowych tak, że

$$f_i^o$$

wewnętrzne zachowanie, odzwierciedlające motywację pieszego
do poruszania się w określonym kierunku z określoną prędkością (wkierunku wyjścia)

$$f_i^{wall}$$

wpływ ścian korytarza na tego pieszego

$$f_{ij}$$

efekty oddziaływania pieszego j na pieszego i.

W tym momencie, poznamy zmianą prędkości v_i pieszego i teraz możemy sformułować całościowe równanie

$$\frac{dv_i}{dt} = f_i^o + f_i^{wall} + f_{ij}$$

2.2.2 Składowe siły 'Social Force'

Łatwo zauważamy, że wykorzystujemy dane eksperymentalne do sprawdzenia poprawności powyższego równania i określenia najważniejszej funkcji interakcji f_{ij}. Przejdźmy przez wszystkie komponenty.

Zachowanie pojedynczego pieszego

Zgodnie z koncepcją 'Social force' przez *Helbing & Molnar 1995* otrzymujemy równanie dla wewnętrznego przyspieszenia $\vec{f_i^o}$:

$$\vec{f_i^o} = \frac{v_i^o e_i^o - v_i(t)}{\tau}$$

Gdy (wszystkie poniższe stałe pochodzą z książki *Helbing 1995*),

$$v_i^o = 1.29 \pm 0.19(ms^{-1}) : \text{pożądane prędkości}$$

$v_i(t)(ms^{-1})$: aktualna prędkość

$\tau = 0.54 \pm 0.05(s)$: czas relaksacji

e_i^o: pożądany kierunek ruchu

Wpływ ścian na tym pieszym

Zgodnie z wcześniejszymi ustaleniami przez *Johansson et al. 2007* takie efekty ścian korytarzy na tym pieszym można opisać za pomocą równania:

$$f_i^{wall}(d_w) = ae^{\frac{-d_w}{b}}$$

Gdy,

$d_w(m)$: odległość prostopadła od pieszego do ściany

$a = 3$ i $b = 0.1$: parametry odpowiadające siłom odpychania tego samego rzędu

Interakcje międzyludzkie

Również zgodnie z poprzednimi badaniami *Johansson et al. 2007*, wiemy, że interakcje międzyludzkie f_{ij} mogą być definiowane jako funkcja odległości i kąta pomiędzy prostymi reprezentującymi kierunki ruchu pieszego i i j. podejścia okazują się jasne i uzasadnione $f_{ij}(d, \theta)$ Daje nam to równanie

$$f_{ij}(d, \theta) = -Ae^{\frac{-d}{B}}(e^{-(n'B\theta)^2}t + e^{-(nB\theta)^2}n)$$

Gdy,

n_{ij} : zmiany kierunkowe jako wektor jednostkowy w lewą stronę $\vec{t_{ij}}$

$d(m)$: odległość między dwoma pieszymi i'em i j'em

$\theta(rad)$: kąt między kierunkiem interakcji a wektorem skierowanym od pieszego i do j

$A = 4.5 \pm 0.3$

$n' = 2.0 \pm 0.1$

$n = 3.0 \pm 0.7$

$t_{ij} = \frac{\vec{D_{ij}}}{\|\vec{D_{ij}}\|}$: kierunek interakcji

Przyjrzyjmy się bliżej parametrowi B: jest zwiększany w kierunku interakcji przez duże prędkości względne i jest zmniejszone, gdy następuje odpychanie w kierunku boków. B zależy więc od:

$$B = \gamma\|D\|$$

Gdy,

$$\gamma = 0.35 \pm 0.01 : \text{parametr równania}$$

I mamy

$$\vec{D_{ij}} = \lambda(\vec{v_i} - \vec{v_j}) + \vec{e_{ij}}$$

$$\lambda = 2.0 \pm 0.2 : \text{względne znaczenie dwóch kierunków}$$

$$\vec{e_{ij}} = \frac{\vec{x_j} - \vec{x_i}}{\|\vec{x_j} - \vec{x_i}\|} : \text{kierunek, w którym porusza się pieszy } j$$

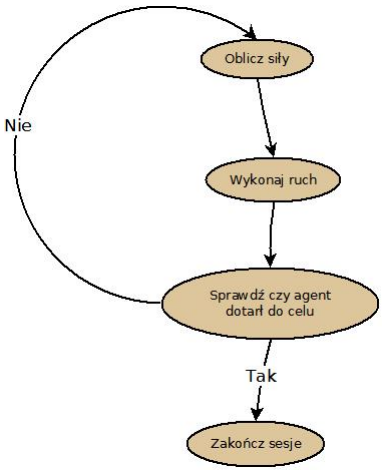
Wykorzystaliśmy początkowo stałe wyznaczone w pracy Helbinga z 2009 roku, później poddaliśmy je kalibracji na własnym modelu.

2.3 Diagramy

2.3.1 Diagramy cykli działań

Agent

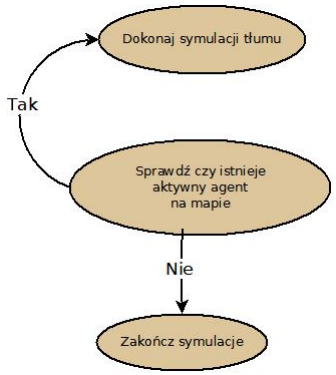
Diagram cykli działań dla jedno agenta



Rysunek 2.1: Diagramy działania dla jednego agenta

Symulacja

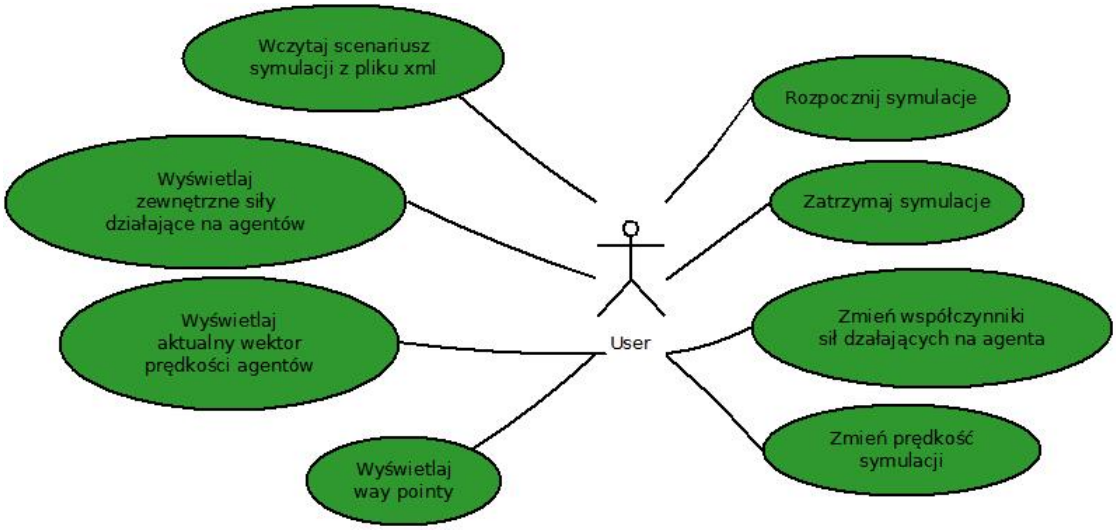
Diagram cykli działań symulacji ruchu pieszych



Rysunek 2.2: Diagramy działania symulacji

2.3.2 Diagramy przypadków użycia

Przypadki użycia systemu symulacji



Rysunek 2.3: Diagramy przypadków użycia systemu symulacji

Rozdział 3

Symulacja komputerowa

3.1 Język i narzędzia programowania

Decyzja dotycząca wyboru platformy językowej była trudna. Pierwotnie mieliśmy skorzystać z takich języków wyższego rzędu jak JAVA, C# ale w trakcie dyskusji o wymaganiach postawionych przed modelem aplikacji zmieniliśmy zdanie. Każdy język ma swoje mocne strony, ale w tym projekcie użyliśmy $C+ + 11$ z następujących powodów:

1. C++ jest językiem kompilowanym, co oznacza, że programy w nim działają bardzo szybko.

2. C++ jest językiem wieloparadygmatowym, więc możemy programować proceduralnie, strukturalnie lub obiektowo.

3. Duża różnorodność bibliotek oraz łatwość ich instalacji.

4. Kompatybilność wsteczna z językiem C.

Ponadto korzystamy z zewnętrznych bibliotek(OpenGl: *glut.h*) do obsługi renderowania symulacji, a także pakietu Qt do stworzenia UI aplikacji.

3.2 Implementacja

3.2.1 Implementacja modelu

Do bezpośredniego renderowania animacji używamy biblioteki glut.h z pakietu OpenGL. Za sterowanie animacją odpowiada klasa QGLWidget dziedzicząca po klasie *QOpenGLWidget*. Aby utworzyć prosty widget animacji, potrzebujemy nadpisać 3 metody:

1. *initializeGL()* - w której inicjalizujemy interfejs za pomocą metod biblioteki glut.h np.:

2. *paintGL()* - w której umieszczamy metody rysujące pojedynczą klatkę animacji; metoda jest wywoływana przez QTimer co każde 10ms,

3. *resizeGL()* - która jest wywoływana przy zmianie rozmiaru okna aplikacji.

Aby biblioteka *glut.h* zaczęła działać musimy ją zainicjalizować w metodzie *main.cpp*

```
int main(int argc, char *argv[])
{
    //glut openGl initialization
    glutInit(&argc, argv);
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Następnie budujemy główny obiekt zarządzający symulowaną sceną SocialForce.h, który inicjalizujemy z pliku XML. Szczegóły budowy w dalszej części. Najważniejszą metodą obiektu SocialForce jest *makeAMove()*, która dostaje *steptime* jak argument i dokonuje obliczenia sił, prędkości i położenia agentów po czasie *stepTime*. Jeśli agent dotrze do celu (ostatniego WayPointa), jest on usuwany ze sceny symulacji.

```
int numberOfAgent = myAgents.size();
for (int i = 0; i < numberOfAgent; i++){
        if (myAgents[i].getID() != -1){
        myAgents[i].makeAMove(myWalls, myAgents, stepTime);
        if (myAgents[i].isReachDestination())
            myAgents[i].setID(-1);
    }
}
```

Dla każdego agenta obliczamy siły w celu obliczenia jego nowej lokalizacji. Do obliczeń użyliśmy wzorów, wypisanych wcześniej w części teoretycznej.

```
forceWithWall = wallInteractForce(walls);
forceByThemselfe = internalForce();
forceWithOthers = agentInteractForce(agents);
acceleration =  forceByThemselfe + forceWithWall + forceWithOthers;

currentSpeed = currentSpeed*0.5 + acceleration * stepTime;

position = position + currentSpeed*stepTime;
```

3.2.2 Interfejs użytkownika

Interfejs aplikacji zbudowany jest w oparciu o bibliotekę *Qt* i środowisko *Qt Creator*, które daje łatwość budowania intuicyjnych aplikacji okienkowych. Interfejs dostarcza użytkownikowi następujące możliwości:

1. wyświetlanie zewnętrznej siły działającej na agentów ze pomocą zmiany kolorów wyświetlania agentów (niebieski - najmniejsza siła, czerwony - największa siła),

2. wyświetlanie aktualnego wektora prędkości aktora,

3. wyświetlania waypoint'ów na scenie,

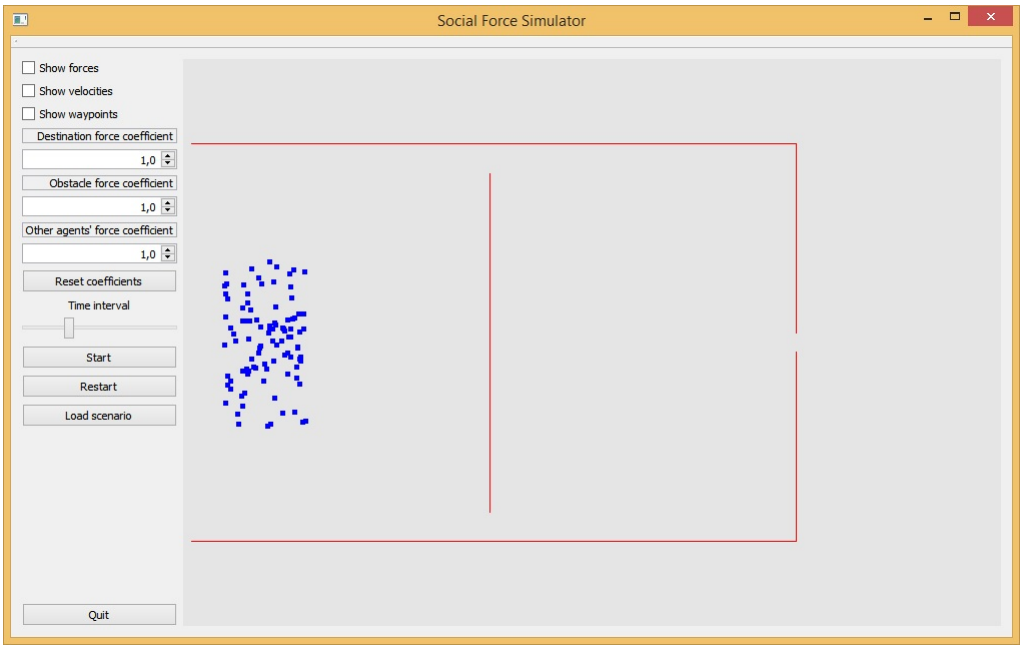
4. kalibracja współczynników sił działających na aktora: siły wewnętrznej, siły oddziaływania z innymi aktorami i siły oddziaływania z przeszkodami(ścianami),

5. zresetowanie współczynników do wartości początkowych,

6. zmiana czasu kroku symulacji jako *stepTime*, dająca możliwość przyspieszania i zwalniania symulacji,

7. zatrzymania/wystartowanie symulacji

8. załadowania sceny symulacji z pliku .xml



Rysunek 3.1: Interfejs użytkownika

3.2.3 Różnice implementacji w porównaniu z wersją książkową

Do obliczania siły interakcji pomiędzy agentem a agentami z otoczenia nie bierzemy pod uwagę wszystkich agentów ze sceny, tylko tych z najbliższego otoczenia. Kolejną zmianą jest to, że do obliczenia sił interakcji uwzględniamy tylko najbliższą przeszkodę, a nie wszystkie znajdujące się na mapie. W wyniku symulacji zmieniliśmy też parametry siły interakcji z przeszkodami na $a = 1.3$ i $b = 0.8$. Symulacje na naszym modelu z książkowymi parametrami nie dawały satysfakcjonującego wyniku. Na agentów działało

zbyt duże przyspieszenie, dlatego dodaliśmy do równania zmiany prędkości parametr α

$$v_{new} = \alpha * v_{actual} + a * t$$

gdzie $\alpha = 0.5$

3.2.4 Inicjalizowanie sceny

Jak już wcześniej wspomnieliśmy, inicjalizowanie sceny odbywa się z pliku w formacie XML.

Cały scenariusz zawiera się wewnątrz znacznika $< scenario > ... < /scenario >$.

Deklaracja waypoint'ów

Najpierw definiujemy waypoint'y, czyli punkty przez które muszą przejść agenci. Waypoint ma *id*, współrzędne x i y, a także promień r.

```
<waypoint id="1" x="-0.92" y="0" r="0.02" />
```

Deklaracja agentów

Definowanie agenta jest trochę bardziej skomplikowane. Każdy agent posiada współrzędne x i y oraz liczbę generowanych agentów n i parametry dx i dy. n agentów jest generowanych ze współrzędnymi początkowymi $(x \pm dx, y \pm dy)$. Dodatkowo dla tych agentów dodaje się waypointy za pomocą znacznika $< addwaypoint.../ >$ z *id* waypointa, które musi się zgadzać, z którymś z waypoint'ów zadeklarowanych wcześniej. Kolejność dodawania waypoint'ów jest istotna; w takiej kolejności będą zmierzali do nich agenci. Całość definicji agenta zamykamy w znaczniku $< agent > ... < /agent >$. Przykład:

```
<agent x="-0.7" y="0" n="100" dx="0.2" dy="0.4">
    <addwaypoint id="2" />
</agent>
```

Deklaracja przeszkód(ścian)

Ostatnim elementem jest zadeklarowanie ścian < *obstacle.../* >. Zawiera on współrzędne *x1, y1* początku ściany oraz współrzędne *x2,y2* końca ściany.

```
<obstacle x1="-0.9" y1="-0.9" x2="-0.9" y2="-0.03" />
```

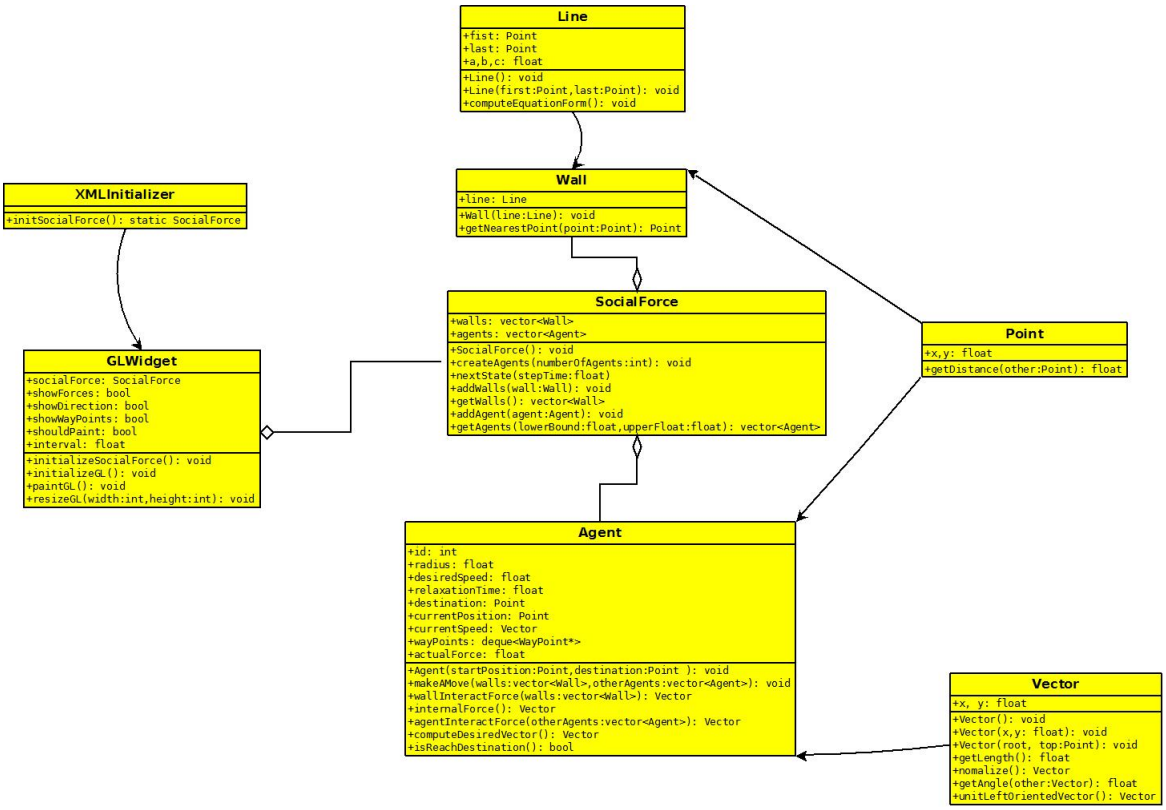
Przykładowa deklaracja scenariusza

```
<scenario>
  <waypoint id="1" x="-0.92" y="0" r="0.02" />
  <waypoint id="2" x="0.92" y="0" r="0.02" />

  <agent x="-0.7" y="0" n="100" dx="0.2" dy="0.4">
    <addwaypoint id="2" />
  </agent>
  <agent x="0.7" y="0" n="100" dx="0.2" dy="0.4">
    <addwaypoint id="1" />
  </agent>

  <obstacle x1="-0.9" y1="-0.9" x2="-0.9" y2="-0.03" />
  <obstacle x1="-0.9" y1="0.03" x2="-0.9" y2="0.9" />
  <obstacle x1="-0.9" y1="-0.9" x2="0.9" y2="-0.9" />
  <obstacle x1="-0.9" y1="0.9" x2="0.9" y2="0.9" />
  <obstacle x1="0.9" y1="-0.9" x2="0.9" y2="-0.03" />
  <obstacle x1="0.9" y1="0.03" x2="0.9" y2="0.9" />
</scenario>
```

3.3 Diagramy klas UML



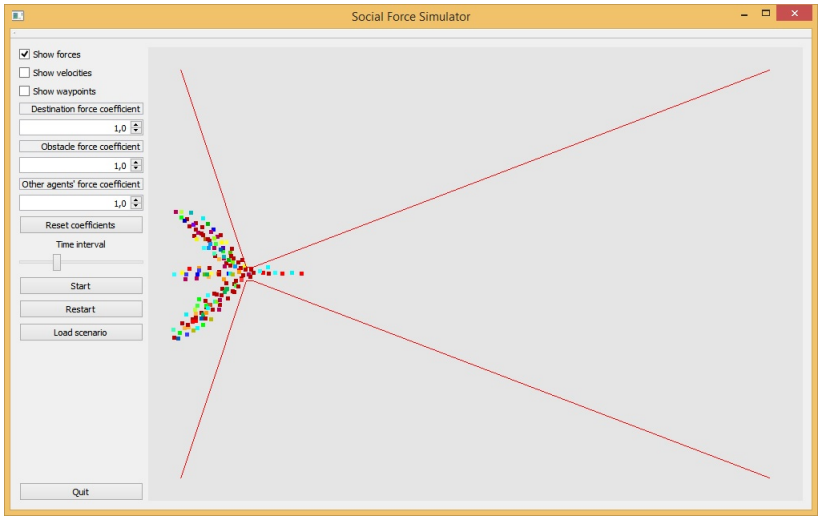
Rysunek 3.2: Diagram klas

Rozdział 4

Wyniki symulacji

4.1 Wyniki i statystyki

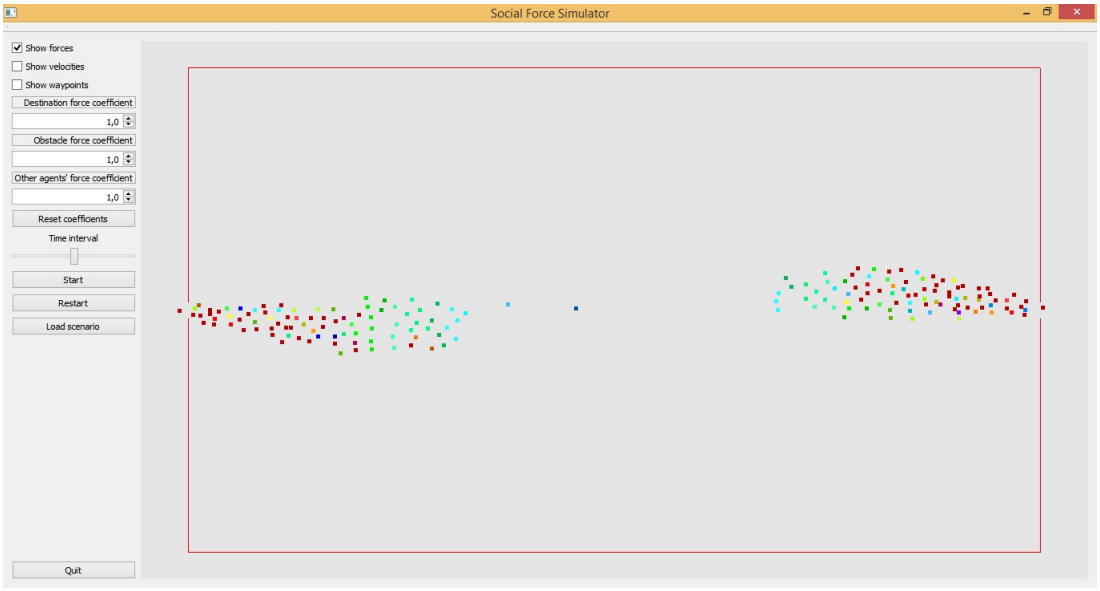
Dzięki przyjętemu modelowi aplikacji łatwe jest zasymulowanie zewnętrznych sił działających na agenta. Widoczne jest to zwłaszcza w miejscach tzw. "wąskiego gardła".



Rysunek 4.1: Wąskie gardło

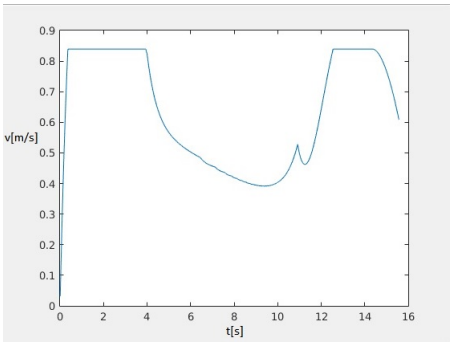
Dane symulacji umożliwiają wykonania różnego rodzaju wykresów np. zmiany położenia pieszych, zmiany prędkości, zmiany sił działających na pieszego w czasie.

Nagła jednostajna prędkość aktora spowodowana jest jest przyjęciem dla



Rysunek 4.2: Przykład sił działających przy wąskim gardle

każdego aktora maksymalnej prędkości ograniczającej.



Rysunek 4.3: Wykres v(t) dla losowego aktora

4.2 Zastosowane procedury kalibracji i walidacji

Korzystając z uzyskanego modelu aplikacji staraliśmy się skalibrować parametry użyte w modelu ruchu pieszych, aby jak najwierniej odzwierciedlały rzeczywistość obserwowalną na co dzień.

Rozdział 5

Wnioski

5.1 Zebranie najważniejszych wniosków

Opracowaliśmy narzędzie programowe zdolne do symulowania i renderowania ruchu dziesiątek a nawet setek pieszych w czasie rzeczywistym. Model Social Force został zaimplementowany w języku C++ z wykorzystaniem zewnętrznych bibliotek glut.h oraz Qt do obsługi interfejsu. Symulacja pokazała, że ruch pieszych można opisać za pomocą prostego modelu siły społecznej dla indywidualnych zachowań pieszych. Pomimo uproszczonego modelu, w zadowalającym stopniu odzwierciedla on rzeczywiste zjawiska zachodzące w ruchu pieszych.

5.2 Zebranie najważniejszych wyzwań i trudności rozpatrywanego problemu

Wystąpiło wiele trudności w realizacji projektu, ale na szczęście znaleźliśmy rozwiązanie (choć nie zawsze idealne). Mieliśmy problem z automatycznym znajdywaniem *waypoint'ów*. Z powodu ograniczonego czasy realizacji projektu, nie udało nam się zaimplementować automatycznego wykrywania takich punktów (np. z wykorzystaniem algorytmu Dijkstry), dlatego przyjęliśmy ręczne ustalanie takich punktów. Innym problemem jest złożoność obliczeniowa, która przy dużej liczbie aktorów powoduje znaczące spowolnienie działania symulacji. Planowaliśmy wykorzystanie wielowątkowości, jed-

nak również i w tym przypadku nie wystarczyło nam wyznaczonego czasu.

5.3 Future Works

Planujemy obecnie zaimplementować algorytmy bardziej skomplikowane w jeszcze bardziej skomplikowanej mapie. Planujemy np. dodanie tzw. *attracting force* jako losowej siły przyciągającej uwagę pieszych czy zaimplementowanie algorytmu Dijkstry jako wybieranie przez pieszego automatycznie najlepszej ścieżki. Dodatkowo chcemy wykorzystać możliwości *multithreading*. Jeśli się nam powiedzie, przyszłym krokiem być może zastosowanie Social Force do opisu procesu formułowania opinii, dynamiki grupy lub innych zjawisk społecznych.

Rozdział 6

Podziękowanie

Projekt modelu Social force jako symulacja dynamiki ruchu pieszych został zrealizowany w ramach przedmiotu "Modelowanie i symulacja systemówńa AGH w Krakowie. Chcielibyśmy podziękować Panu Dr. hab. inż. Jarosławowi Wąs za cenne i inspirujące uwagi i instrukcje.

Bibliografia

[1] Helbing D., *Complex System*, Germany (1992).

[2] Mehdi Moussaıd , Niriaska Perozo, Simon Garnier, Dirk Helbing, Guy Theraulaz, *The Walking Behaviour of Pedestrian Social Groups and Its Impact on Crowd Dynamics*, (2010).

[3] FDS+Evac, Timo Korhonen, Simo Hostikka, *Fire Dynamics Simulator with Evacuation*, (2009).

[4] Dijkstra J., Jessurun A., Timmermans H., *A multi-agent cellular autotmata model of pedestrian movement*, Pedestrian and Evacuation Dynamics, Berlin (2000).

[5] Helbing D., Molnar P., *A social force model for pedestrian dynamics*, Phys. Rev. E 51 (1995).

[6] Lohner R., *On modeling of pedestrian motion*, Applied Mathematical Modeling doi:10:1016 (2009).

[7] Mehdi Moussaıd, Dirk Helbing, Simon Garnier, Anders Johansson, Maud Combe, Guy Theraulaz„ *Experimental study of the behavioural mechanisms underlying self-organization in human crowds*, (2009).