Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

# Learning Crowd Behaviour with Neuroevolution

Master's thesis

Pascal Widmer

pawidmer@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

**Supervisors:**
Manuel Eichelberger, Michael König
Prof. Dr. Roger Wattenhofer

May 19, 2017

# Abstract

Many different techniques are used to mimic human behaviour in order to create realistic crowd simulations. Agent-based approaches, while having the most potential for realism, traditionally required carefully hand-crafted rules. In recent years the focus has shifted from hand-crafting decision rules to learning them through methods such as reinforcement learning. In this work a closer look is taken on the suitability of a prominent neuroevolution method called NeuroEvolution of Augmenting Topologies (NEAT). Agents are controlled using an artificial neural network, which is evolved over generations in typical crowd simulation scenarios. The evolved control logic is then replicated to many agents and the emergent crowd behaviour is empirically evaluated.

# Contents

Chapter 1

# Introduction

Large-scale crowd simulations have become an integral part of many recent films and computer games. Traditionally, each agent required to be animated by hand, increasing the cost of production. Crowd simulation tools aim to alleviate this problem. They provide ways for artists to control the movement of groups of agents, as well as letting them specify behavioural rules for specific agents [1]. However, group-based techniques often lead to homogeneous crowds with little diversity. Agent-based rules, on the other hand, need to be carefully hand-coded so as to both create realistic local behaviour and to satisfy the desired group dynamics on a larger scale. Machine learning can serve as a means to learn agent-based behavioural rules automatically while satisfying global constraints.

Neuroevolution has been shown to outperform Q-learning on tasks such as single pole balancing and similar control tasks [2]. Neuroevolution of Augmenting Topologies (NEAT) and similar neuroevolution techniques have the advantage of searching directly in the behavioural space for good solutions as opposed to iterating over value functions. Because neuroevolution keeps a population of candidate solutions which potentially solve the same problem in unique ways, it has the additional benefit of adding diversity for free. NEAT seems to be a natural contender for tasks that have continuous high-dimensional state spaces and where interesting but imperfect solutions are not undesirable.

In neuroevolution, agents are controlled through artificial neural networks which are iteratively improved. This is done by specifying how agents may interact with the environment and what their global goal is. Agents are evaluated after each episode of interactions and their performance is measured using a fitness function. Typically, the more successful agents are mutated by randomly tweaking the artificial neural network connection weights, while worse performing individuals are discarded. This naturally leads to an improvement of the population. Fully evolved agents react locally to the environment and as a result of being trained on a global fitness function they satisfy the global dynamic constraints.

## 1.1 Related Work

Similar work has been done by Sunrise Wang et al [3]. They compare how Conventional Neuro-Evolution (CNE), Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES), Neuro-Evolution of Augmenting Topologies (NEAT), and Enforced Sub Populations (ESP) perform on various multi-agent learning tasks. A lot of research, on the other hand, implements some form of single-, or multi-agent reinforcement learning instead. Francisco Martinez-Gil et al [4] use vector quantization (VQ) to generalize the continuous state space and apply multi-agent reinforcement learning to a pedestrian navigation scenario. Similarly, L. Casadiego and N. Pelechano [5] use reinforcement learning on a single agent in an obstacle avoidance scenario. Later, the learned table is shared among many agents to create a crowd.

Chapter 2

# Background

## 2.1 Artificial Neural Network (ANN)

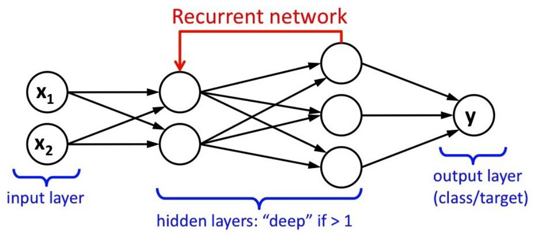
An *artificial neural network (ANN)* is a network of neurons loosely mimicking natural neurons found in a brain. Figure 2.1a shows the structure of a simple ANN. The network can be subdivided into three layers of neurons. The input layer, which receives sensory input from the environment, the output layer, whose neuron's output can represent classifications or actions, and the hidden layer or layers in between. Input values are propagated through the network over the edges and each neuron applies an activation function to its input. More concretely, the input to each neuron $j$ is the weighted sum of its in-edges, i.e., the so called *transfer function*

$$\theta_j = \beta_j + \sum_{i=1}^{n} x_i \cdot w_i$$

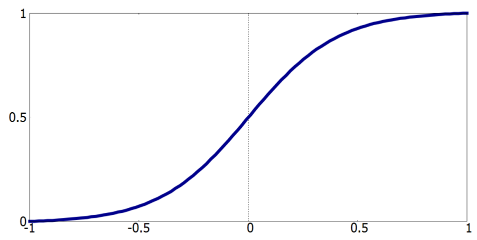
where $x_i$ represents the value travelling over edge $i$, $w_i$ the weight associated with the edge and $\beta_j$ the bias associated with neuron $j$. The *activation function* is typically a logistic sigmoid given by

$$f(\theta) = \frac{1}{1 + e^{-\beta\theta}}$$

and illustrated in Figure 2.1b, in this case with slope parameter $\beta$ equal to 4.9. It maps $\theta$ to values in the interval $(0, 1)$. Many non-linear activation functions can be used instead, however sigmoids are most common. ANNs which contain cycles are called recurrent and give rise to *recurrent neural networks (RNNs)*. They have the capability to exhibit dynamic temporal behaviour and are particularly suited for tasks which require some form of memory. ANNs serve as the control logic of agents in NEAT. Often times, neurons do not have an associated bias value but instead the bias is modelled as an additional input to the network with a constant value of 1. This way, whenever a neuron needs a bias value an edge is simply added between the constant input neuron and the edge weight is used to scale it appropriately. Although it is mathematically equivalent it helps reduce



(a) Artificial Neural Network (ANN) [6]



(b) Sigmoid activation function $f(\theta) = \frac{1}{1+e^{-4.9\cdot\theta}}$

Figure 2.1

the computational cost in NEAT, as having a bias value for every node would increase the search space unnecessarily.

## 2.2 Evolutionary Algorithms (EA) and Neuroevolution (NE)

*Evolutionary algorithms (EA)* aim to find solutions or good approximations to optimization and search problems by mimicking the way natural evolution works. Randomly generated candidate solutions (not necessarily ANNs) represent individuals in a population. Similarly to natural evolution, a selection pressure is applied to these individuals based on their fitness, i.e., a measure of how well they solve the given problem. Better performing individuals having achieved a higher fitness are then recombined (a process called crossover) or randomly mutated, while worse performing individuals are removed from the population. The population's mean fitness thus improves in each generation. There is generally no guarantee that a global optimum is found, however there exist techniques such as speciation to ensure that the population stays diverse enough to explore a large part of the problem space. When individuals in an evolutionary algorithm are represented as an ANN, the algorithm is called *neuroevolution*.

## 2.3 NeuroEvolution of Augmenting Topologies (NEAT)

When neuroevolution is used to evolve both weights and topology of an ANN it is referred to as TWEANN (Topology and Weight Evolving Artificial Neural Network). One of the most popular such algorithms is *Neuroevolution of Augmenting Topologies (NEAT)*, introduced by Kenneth O. Stanley and Risto Miikkulainen in 2002 [7].

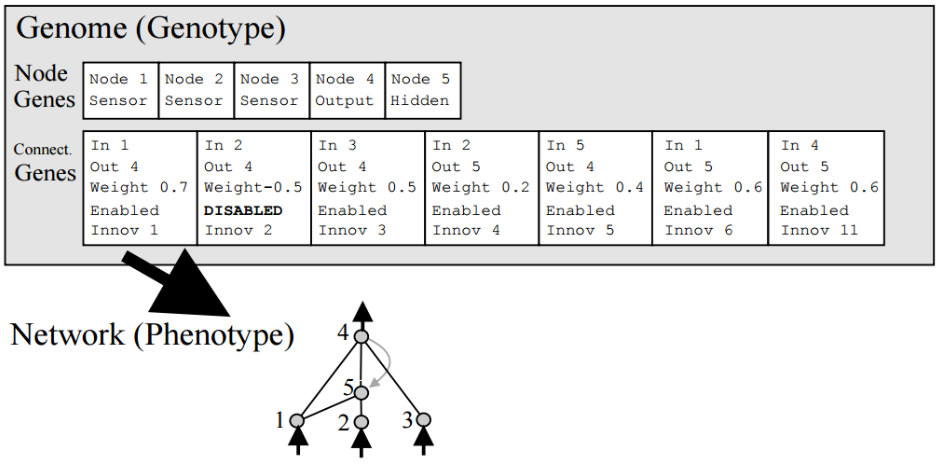

Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Output
Node 5 Hidden
Connect. Genes
In 1 Out 4 Weight 0.7 Enabled Innov 1
In 2 Out 4 Weight-0.5 DISABLED Innov 2
In 3 Out 4 Weight 0.5 Enabled Innov 3
In 2 Out 5 Weight 0.2 Enabled Innov 4
In 5 Out 4 Weight 0.4 Enabled Innov 5
In 1 Out 5 Weight 0.6 Enabled Innov 6
In 4 Out 5 Weight 0.6 Enabled Innov 11
Network (Phenotype)
4
5
1
2
3

Figure 2.2: Genetic encoding of an ANN in NEAT [7]

### 2.3.1 Genetic Encoding

In NEAT ANNs are encoded genetically, i.e., as a sequence of genes an example of which is shown in Figure 2.2. *Node genes* specify whether nodes are of type sensor (input), output or hidden. *Connection genes* specify which neurons are connected and the respective edge weight. Connection genes can be disabled and re-enabled in later generations, akin to how natural DNA segments can be recessive and only expressed in later generations. Connection genes also carry a global *innovation number* or *history marker*. When a new connection gene is introduced by a structural mutation it is assigned a new and globally unique innovation number. If the connection gene is inherited from a parent it keeps the parent's innovation number. Innovation numbers can thus be used to tell whether connection genes have the same ancestor (are homologous) or whether they were introduced through a structural mutation.

### 2.3.2 Mutations and Crossover

**Mutations** The evolution of individuals happens through either mutations or crossover. Mutations include structural mutations as depicted in Figure 2.3, namely adding a new connection to the network and adding a new node to the network. Adding a new connection involves inserting a new connection gene ($3 \rightarrow 5$) and assigning it a new innovation number (7). When a new node is introduced a new node gene is inserted. If a direct connection between the nodes has already been present ($3 \rightarrow 4$) the according connection gene is disabled (DIS) and two new connection genes are introduced ($3 \rightarrow 6$) and ($6 \rightarrow 4$), each of which is assigned a new innovation number (8 and 9). The third and most typical mutation is a simple edge weight mutation. Each edge weight has a chance to be assigned a new random value or be slightly perturbed (sampled from a Gaussian distribution).

**Crossover** In a process called *crossover* two individuals of the population are recombined inheriting genes from both parents. Figure 2.4 shows a possible crossover scenario. The list of connection genes of both parents are compared using their innovation numbers. When innovation numbers match genes are inherited randomly from either parent, whereas disjoint and excess genes are inherited from the fitter parent. Disjoint and excess genes are genes not present in both parents, excess genes are mismatches in the end of the genome while all other mismatches are disjoint genes. In the special case, where parents have equal fitness, as is the case in the given example, genes are inherited from both parents. Genes 6,7,9,10 are inherited from parent 2 whereas gene 8 is inherited from parent 1. All inherited genes also have a slight chance of becoming disabled and disabled genes have a chance to become re-enabled.

Offspring born through crossover often have the potential to increase the fitness of their parents by a large amount, as they might have inherited topologies which solve two different sub-problems. Initially the offspring's fitness might be lower because evolution needs to first optimize the network to combine the two inherited topologies. In general, larger networks take a longer time to optimize as more parameters have to be mutated. To prevent new offspring from being prematurely removed from the population the concept of species is introduced.

### 2.3.3 Speciation

In NEAT individuals of a population are distributed into separate species on the basis of their similarity. Individuals of species compete among each other for survival as opposed to the population at large. Innovation numbers can again be used to measure topological similarity between two individuals. The similarity or distance $\delta$ between individuals is a linear combination of the number of excess genes $E$, the number of disjoint genes $D$, and the average weight difference for matching genes $\bar{W}$ given by

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W}.$$

$c_1$, $c_2$, and $c_3$ are used to define the importance of the three factors. NEAT uses a concept called *explicit fitness sharing* to ensure that no species can take over the whole population. In explicit fitness sharing an individual's fitness is divided by the number of individuals inside the same species.

On one hand speciation allows offspring generated through crossover to have a chance to optimize. On the other hand it allows simpler networks to stay in the evolutionary race without being replaced by larger networks.

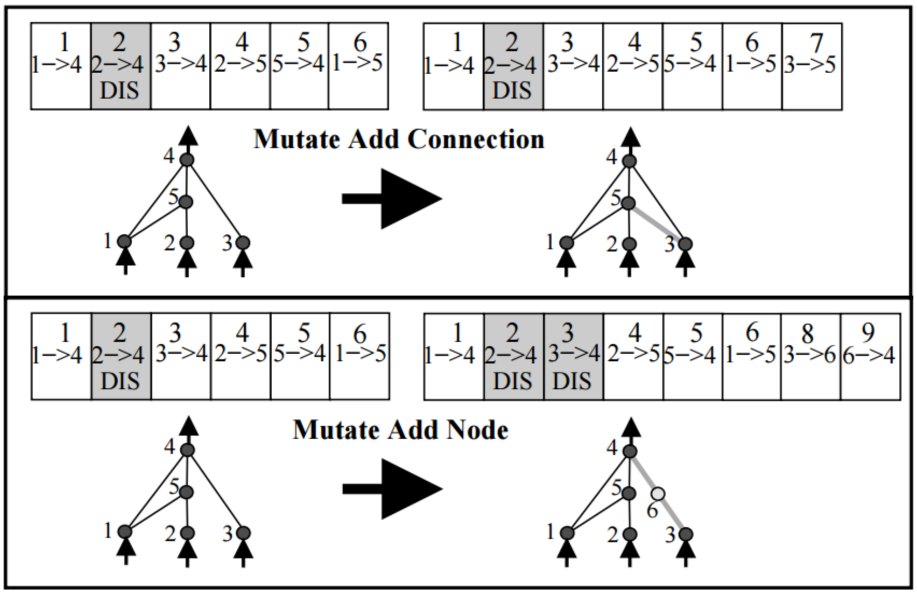


Figure 2.3: Structural mutations in NEAT [7]

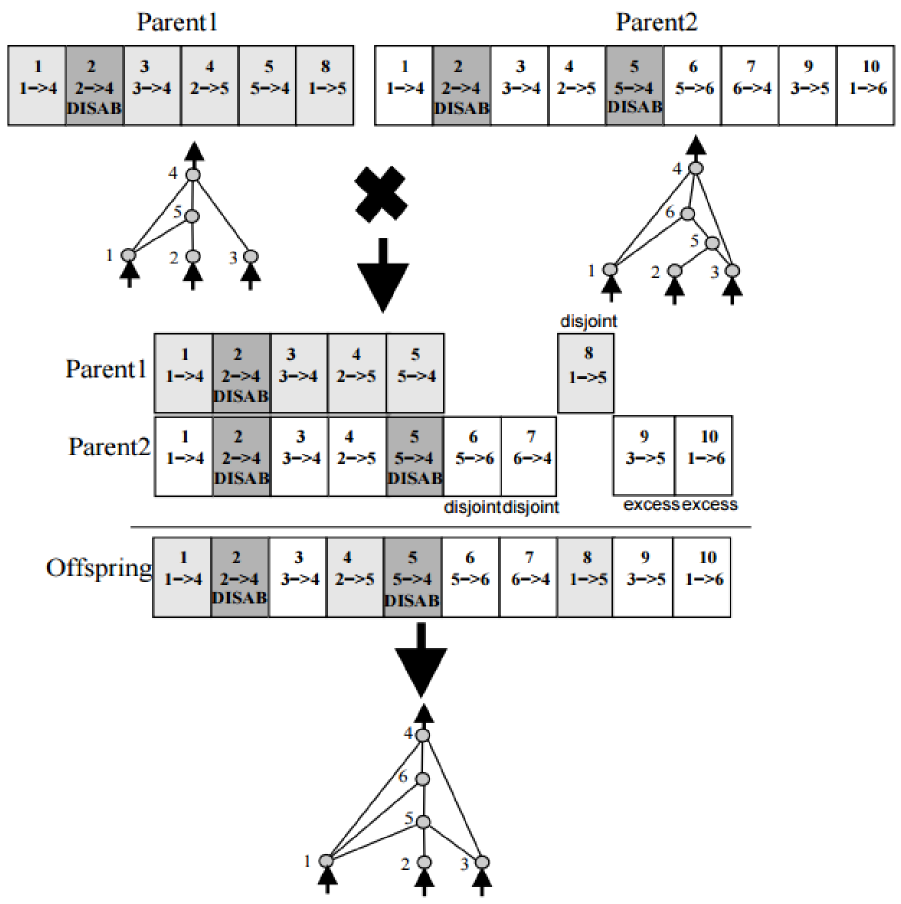


Figure 2.4: Crossover in NEAT [7]

Chapter 3

# Implementation

For NEAT to succeed in evolving an artificial neural network that solves a given problem it requires many evaluations of each individual in the population. Box2DX [8] is a C# port of Box2D (C++) and is used as a fast 2-dimensional physics simulator. A population is trained in the Box2D environment and rendered using OpenGL when needed. SharpNEAT [9] implements NEAT and Maya 2017 [10] is used to render the final crowd simulation.

## 3.1 Evolving Agent Behaviour

Agents are trained alone or in a small groups. When a small group of agents is trained their ANN is identical, however their environment now consists of other agents which enables them to interact with each other. Interaction is needed for tasks with collaborative goals, but comes with the downside of much slower Box2D evaluations and thus slower evolution.

**Agent Model** Depending on the scenario agents receive different inputs from the environment. Inputs include vision sensors, hearing sensors, communication channels and proximity sensors. All inputs are scaled to the interval [0,1] or [-1,1], as required by NEAT, and fed to agents on a per-frame basis. Likewise, the ANN or ANNs are evaluated every frame and the output is applied to agents every frame. Usually output from the ANN consists of a torque value [-1,1] and a linear force [-1,1], which is applied in the direction the agent is facing. Both values are up-scaled for realistic movements and the linear force is halved for backwards movement. The Box2D environment applies linear and angular damping to agents to limit their maximum angular and linear velocities, increasing realism. In some scenarios more sophisticated agent models were implemented, especially in regards to motion. For example, in one attempt agents were able to turn their head in addition to moving their body. Because NEAT did not properly learn to make use of them they are not part of the final work.

**NEAT Parameters** Throughout most scenarios the following settings were used for NEAT, as they proved to be most successful.

***Population size (300)*** The fixed number of individuals in the population.

***Number of species (30)*** The maximum number of species, which are allowed to exist at any given point in time.

***Neuron activation function (Steepened sigmoid)*** The activation function applied by the artificial neurons. More concretely, $f(\theta) = \frac{1}{1+\epsilon^{-4.9 \cdot x}}$.

***Activation scheme (Cyclic fixed iterations 1)*** The activation scheme describes how the neural network is structured and evaluated. Cyclic means the network is a RNN and can contain cycles. Each ANN connection is traversed exactly once in the *fixed iterations 1* setting. It means that input values might not immediately affect the agents behaviour as there might not be an edge connecting an input neuron directly to an output neuron. This configuration fits the per frame continuous feed from the environment well. Because 30 frames per second are used to simulate the environment, an agent's input barely changes between frames.

***Complexity Threshold (100)*** The maximum number of edges in the evolved ANN. ANNs are pruned when they reach the complexity threshold.

***Weight range ([-5, 5])*** Edge weights can only take on values in this range.

***Mutation (0.5)*** The offspring are generated through mutations with probability 0.5 and through crossover with probability 0.5.

***Connection weight mutation probability (0.988)***

***Add neuron mutation probability (0.001)***

***Add connection mutation probability (0.01)***

***Delete connection mutation probability (0.001)***

## 3.2 Crowd Simulation

When a single agent or a group of agents achieves the desired fitness the ANN is extracted and saved in a text file. A new and potentially larger Box2D environment is created, which reads the file, loads the ANN description and duplicates it to any number of agents. Alternatively, the entire population of ANNs or the better performing ones can be exported and loaded again. This leads to additional diversity in the crowd. Diversity also arises naturally from the agent's local view of the environment. Different experiments can be combined to create

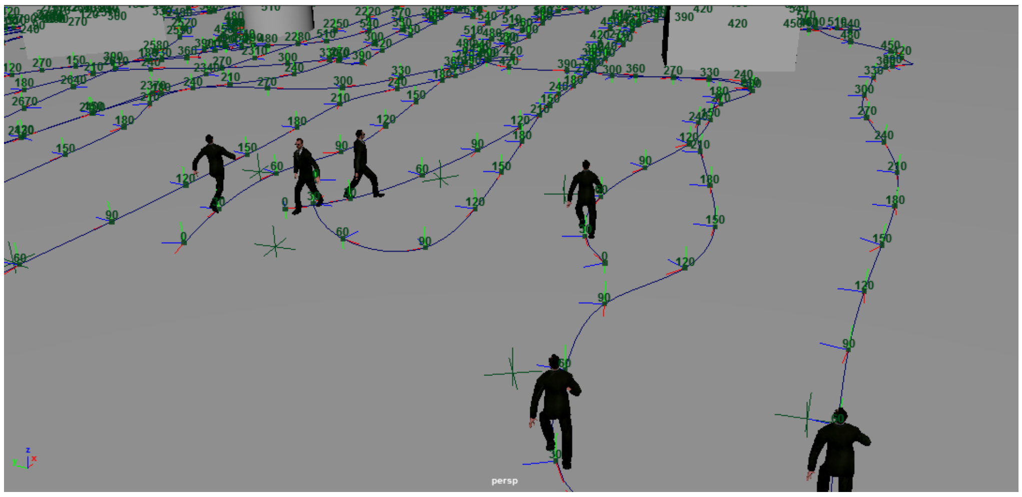


Figure 3.1: Motion paths define the path for each agent, including position markers (numbered) and orientation markers every 30th frame.

more sophisticated scenarios where agents are controlled by different ANNs. In a final scenario, which is rendered in Maya, three scenarios are combined and each agent's behaviour is determined by his location by seamlessly exchanging the ANNs.

## 3.3 Visualization in Maya

In a final step the Box2D crowd simulation trace is exported to a text file. Each object's and agent's position, orientation and velocity is saved periodically. A script written in Maya Embedded Language (MEL) parses the text file and recreates the environment in Maya 2017. Agents are represented using a model of Agent Smith [11] from the movie The Matrix. A number of motion capture clips are applied to Agent Smith using Maya's Quick Rig tool and converted to in-place animation clips. They include walk cycles, slow and fast run cycles and a transition from standing to running. The agent's motion trace is represented using a motion path and the agent's pivot point is attached to it. Position markers as well as orientation markers on the motion path are used for fine-grained control over the agent's velocity and orientation as seen in Figure 3.1. The agent itself is animated using a blend of the animation clips. They are dynamically cycled and aligned to roughly fit the agent's speed. See Figure 3.2 for an example. Static objects are created, translated, rotated and scaled while cars from the Road Crossing Scenario are animated using keyframes and interpolated linearly. A model of the AC Cobra MKII [12] is used for cars. A scene of the final rendering is depicted in Figure 3.3.

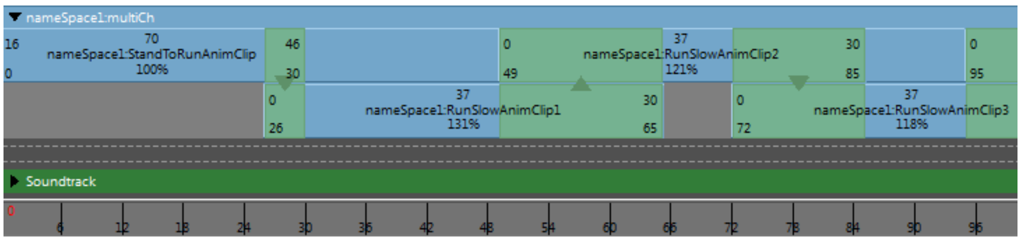


Figure 3.2: Animation clips are blended dynamically to create believable movement.

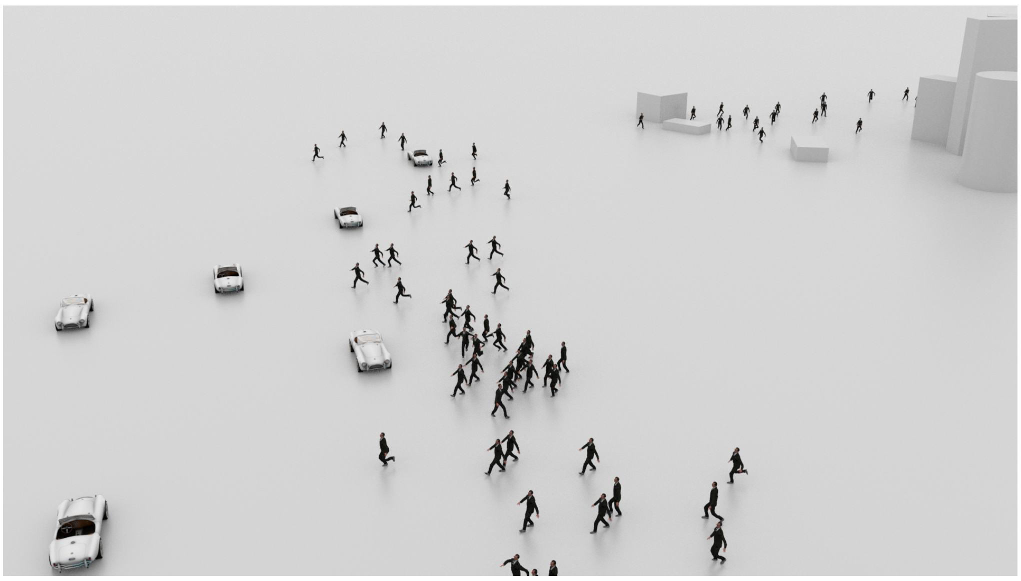


Figure 3.3: A scene from the final Maya rendering.

Chapter 4

# Scenarios

Several crowd simulation scenarios have been identified and implemented. They include tasks such as avoiding obstacles, pathfinding, learning right-hand traffic (or left-hand) to efficiently pass through tight choke points, evading aggressive clowns, avoiding getting hit by a rotating bar, finding objects by moving randomly, crossing a road while avoiding cars, finding food collaboratively and learning social behaviour in a school setup. Some of them have been inspired by previous work done as part of a bachelor thesis on crowd simulations by C. Maurer [13].

## 4.1 Road Crossing

**Setup** In the road crossing environment shown in Figure 4.1 agents need to cross a sequence of perpendicular roads without hitting any car. While roads have spaces between them, cars can travel in both directions with different velocities. Each agent knows the distance to the edge of the next road, the distance to the next oncoming car on this road (as if they heard it) and whether their target location is on their left or right. Agents spawn randomly in the area below all roads and cars spawn in random locations on roads initially. When cars reach the edge of the simulated environment they reappear on the opposite side again. The fitness of each agent depends on how far they get to the target location and whether they have collided with a car.

**Result** How often agents reach their target location generally depends on how fast agents can run away in relation to the car's speed. Agents learn to avoid oncoming cars by either backing off or by travelling in parallel to the roads until the traffic has cleared. The learned behaviour is not perfect and some agents still collide or touch cars on their way to the target location.

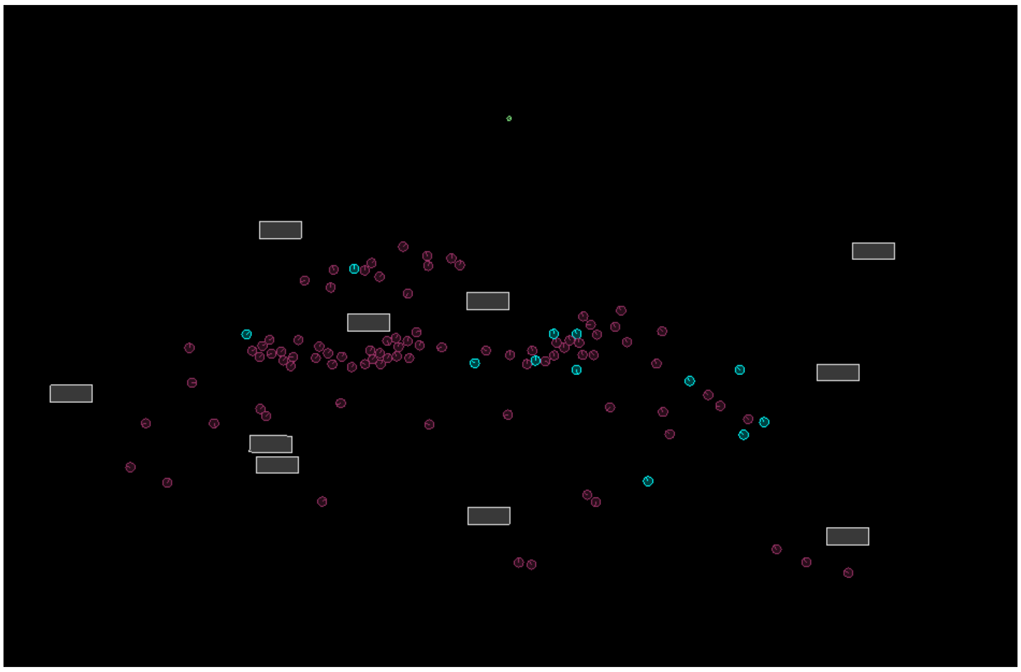


Figure 4.1: Road Crossing Scenario - The green circle on top represents the target location. Teal colored agents have collided with or touched cars (rectangles).

## 4.2 Collaborative Food Gathering

**Setup** To investigate whether more complex social behaviour can automatically be learned, a scenario which requires some form of collaboration is implemented. The goal of agents is to gather as much food as possible before the time runs out. Food can only be consumed when some fixed number of agents are close by, hence they need to move in groups or call each other upon finding food. In the Collaborative Food Gathering Scenario ten agents are trained together while five are needed to consume food. Consuming food increases each agent's energy, which in turn determines how long they live. Every second that passes decreases an agent's energy. Agents have a vision sensor which tells them whether there is food in front of them, depicted in Figure 4.2 as a grey sector of a circle. Apart from acceleration and steering, each agent has an additional output which can be used to communicate by sending 0 or 1. Agents hear the closest agent that sent 1 and receive the distance and angle to this agent as input. A proximity sensor tells them the number of nearby agents in a fixed radius around them and a random number input can be used to create random walks.

**Result** The resulting learned behaviour consists of a first phase in which all agents periodically send 0 or 1. Because agents learn to head towards the closest

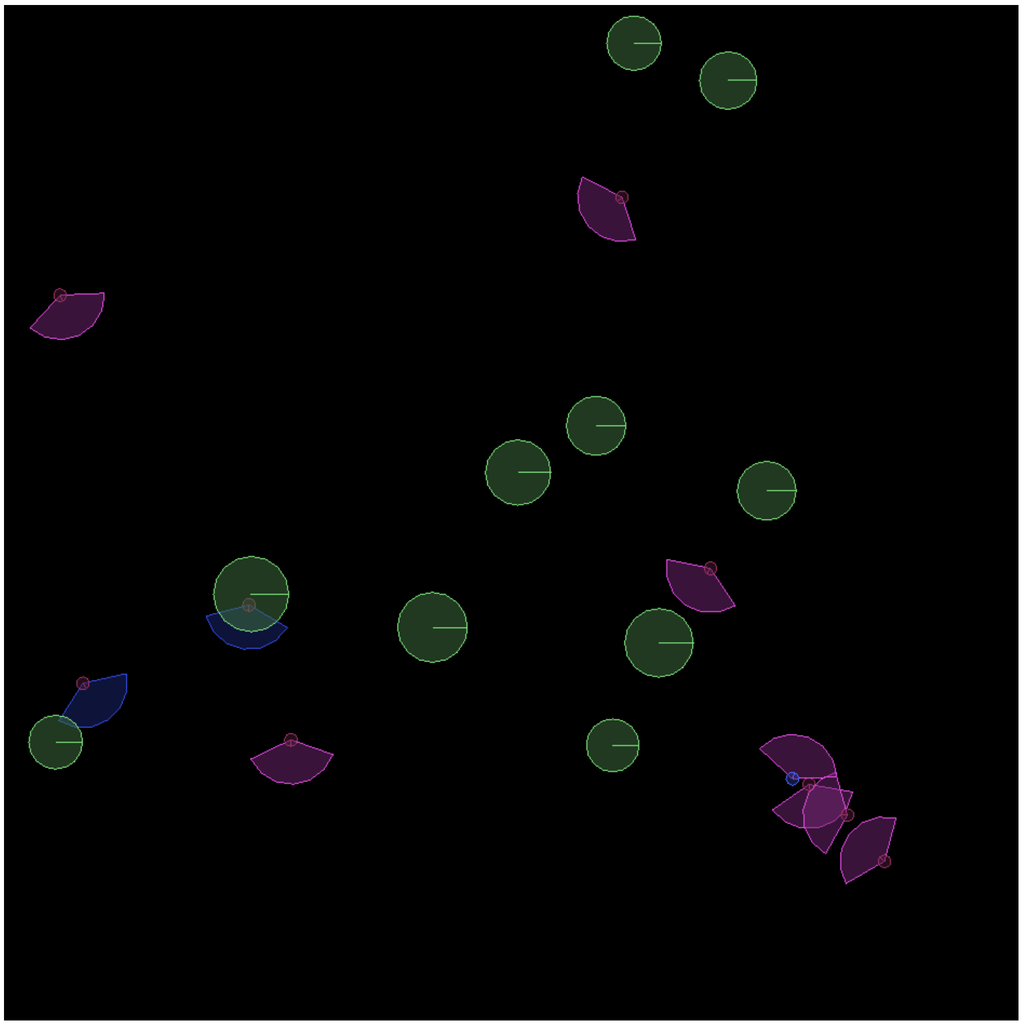


Figure 4.2: Food Gathering Scenario - Green circles represent food. Agents have an attached vision sensor represented as a sector of a circle. When food is detected the sensor turns purple, otherwise it is blue. When an agent sends 1 the agent's body turns blue, otherwise it is red.

calling agent they naturally converge to the center of mass and a group is formed. The agents then slowly wobble around the group's center and occasionally find food nearby. This behaviour is far from optimal, as the group only travels very slowly, if at all. A more efficient solution would be to simply follow a leader on his random walk.

In a similar set-up, agents are given the distance and angle to the closest food and their food sensor is removed, effectively getting rid of the searching phase. In this scenario, agents still form a group but the group can gather food fairly efficiently. Initially, agents alternate between heading to nearby food and heading to the closest calling agent and as a result the group travels. In later generations a leader begins to emerge which guides the agents. Initially, the leader is frequently exchanged by another agent from the group. Further evolution increases the time between those transitions and as a result the group gathers food quicker.

## 4.3 School Domain

**Setup** The school domain has been inspired by Lisa Torrey's work [14], which investigates the use of multi-agent reinforcement for crowd simulations. In the school domain, students are spawned randomly in the hallways of a school and need to reach their assigned classroom when the bell rings. On their way to their classroom they may socialize with other students. Five students are spawned randomly in the hallways. All students have preassigned rooms where they need to be when the bell rings. They have a limited cone of vision which tells them the number of students in front of them. In addition, they know the time left to reach their classroom, the distance to the classroom and where it is located as an angle relative to their current direction. The fitness function is defined to be zero if students do not reach their classroom on time and if they do it is equal to the time spent socializing plus a fixed reward for reaching the classroom.

**Result** Students learn to change their behaviour according to the time left. Initially, they start moving straight to their classroom but as soon as they see students nearby, they stop and wait or continue very slowly. When the time reaches a certain threshold, they stop socializing and run to their assigned classroom as intended. Students could in theory achieve a higher fitness had they learned the time-distance dependency. This never happened however, partly because the few additional seconds of socializing has little effect on the fitness and because spawning in random locations in the hallway results in high fitness fluctuations which obscure the potential fitness gains.

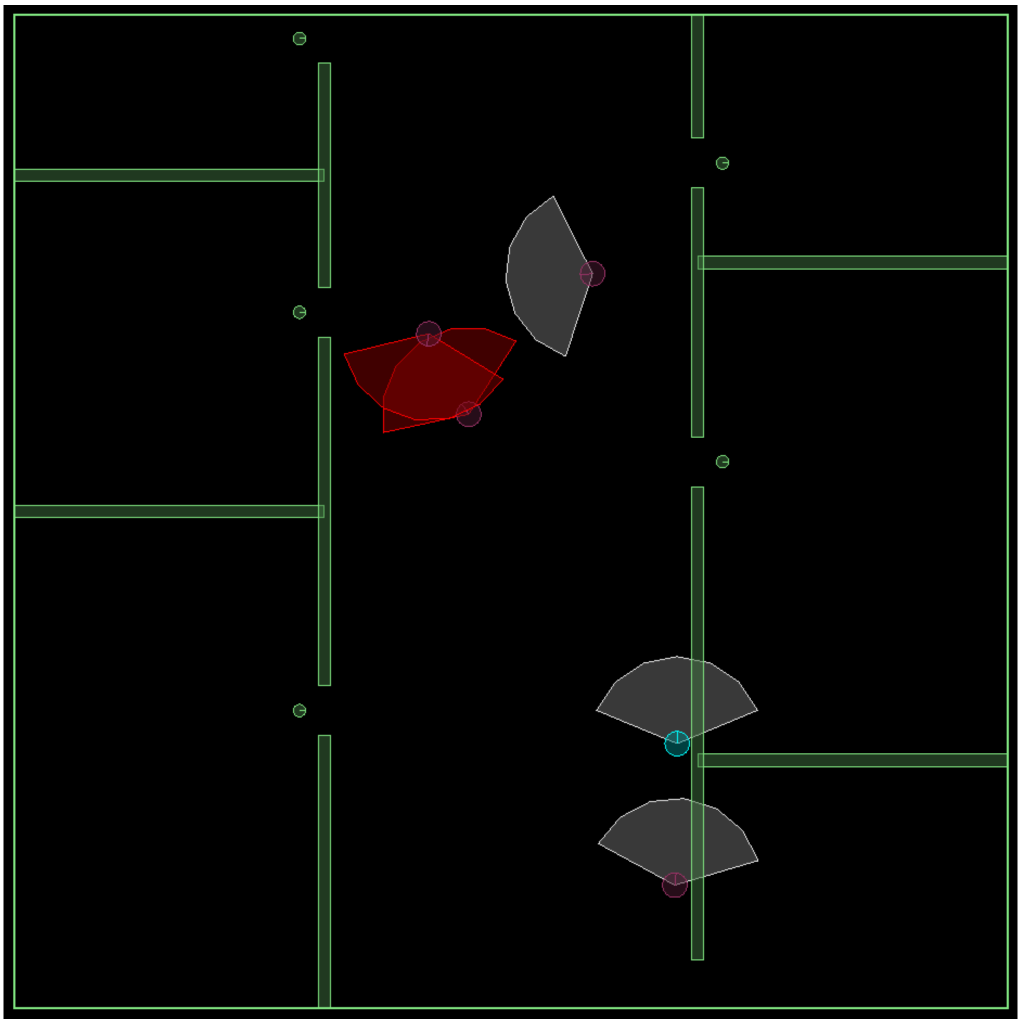


Figure 4.3: School Scenario - The green circles represent target locations. An agent's sensor turns red when it detects other agents. An agent's body turns from red to teal when it collided, however, this is not part of the fitness function in this scenario.

Chapter 5

# Evaluation

## 5.1 Performance

Box2D evaluations are done using multiple threads in parallel without rendering the scene. Every second, the physics engine can calculate around one million physics steps containing a single agent. Scaling up the number of agents drastically decreases performance as both the number of collisions increase and because a force is applied to each agent every frame (in a 30 FPS setting). Each fitness evaluation consist of roughly 3000 frames (100 seconds) leading to around 100 fitness evaluations per second. For NEAT to learn rudimentary behaviour a few minutes suffice while learning more complex behaviour generally takes hours. Recreating a scene with 100 agents in Maya takes a minute, while rendering a 2 minute video sequence takes many hours.

## 5.2 Extensibility

Adding new experiments involves adding an entry to the list of experiments for NEAT. Among other things, it describes the number of inputs and outputs to the ANN and where to find the appropriate class which handles the evaluation logic. The Box2D environment is completely described in an external text file. Upon starting a new experiment and before each evaluation the Box2D world is loaded and created dynamically. The description consists of all the position of objects, their behaviour, including the number of agents and their input sensors. Some minimal coding is required to map the agent's output and input to the ANN's input and output and to specify the fitness function.

## 5.3 Comparison to Reinforcement Learning

A natural question to ask is whether NEAT can compete with state of the art reinforcement learning (RL) approaches on the presented crowd simulation tasks.

In the obstacle avoidance scenario both NEAT and an actor critic method reinforcement method are applied. The implemented RL method is part of the library dotRL written by RLBartosz Papis et al [15]. The scenario involves guiding an agent around multiple static objects to the target. In the case of reinforcement learning, the state of the environment is described by the input of the five vision sensor, the agent's distance from the target and angle to the target, his angular velocity, and the linear velocity vector. Whereas NEAT learns after every complete agent evaluation, RL learns from each sample every frame. Because of this, it is natural to alter the fitness function for RL. In RL, the fitness for a single action is given by the covered distance to the target. Similar to NEAT, a few hundred evaluations are enough to learn to move towards the target location. However, the RL method quickly teaches an agent to spin around himself without moving towards the target location soon after. Because of time constraints and poor documentation of dotRL the reason for this behaviour could not be determined.

Chapter 6

# Conclusion

NEAT solves the devised tasks fairly fast and to a degree which suffices for many crowd simulation applications. Nevertheless, the population's average fitness barely improves at some point even though better solutions exist. Part of the reason is most likely that NEAT needs to find a solution which is generalizable. Experiments include randomly placed objects and random starting locations, which are designed to change between each fitness evaluation. The reason being, that in the final crowd simulations agents need to be adaptable, they might find themselves in similar but not identical situations. Adding randomness makes it less clear whether a given network solves a task better or not. Additionally, it proved hard to evolve an ANN that could properly store long term memories. Fortunately, most crowd simulation tasks do not require extensive use of memory.

The model used for agents, in terms of environment sensors and the possible movements, is deliberately chosen to be simple. Increasing the number of inputs from the environment would in theory encourage the evolution of a more intelligent decision logic but in practice it simply leads to slower evolution in NEAT where the additional inputs are never utilized. This becomes evident when thinking of the number of potential ANN topologies in relation to the number of input nodes. Often, a more complex decision logic has only a small effect on the achievable fitness and as such, the behaviour is hard to distinguish from noise caused by random environments. On the other hand, it makes sense striving for a simple model of an agent that can be reused in different set-ups.

One of the advantages of NEAT is its ability to evolve sometimes unpredictable and interesting behaviour. In the Road Crossing Scenario, for example, agents learn to employ different evasion strategies. While most agents patiently wait for the passing car some decide to run along the road and others simply back off. In some cases, minuscule differences in the ANN's inputs lead to completely different decisions. In large scale crowd simulations this stochastic behaviour seems to create a believable and diverse crowd.

# Bibliography

[1] Thalmann, D., Grillon, H., Maim, J., Yersin, B.: Challenges in crowd simulation. In: CW. (2009)

[2] Moriarty, D.E., Miikkulainen, R.: Efficient reinforcement learning through symbiotic evolution. Machine Learning (AI94-224) (1996) 11–32

[3] Wang, S., Gain, J.E., Nistchke, G.S.: Controlling crowd simulations using neuro-evolution. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. GECCO '15, New York, NY, USA, ACM (2015) 353–360

[4] Martinez-Gil, F., Lozano, M., Fernández, F. In: Multi-agent Reinforcement Learning for Simulating Pedestrian Navigation. Springer Berlin Heidelberg, Berlin, Heidelberg (2012) 54–69

[5] Casadiego, L., Pelechano, N.: From one to many: Simulating groups of agents with reinforcement learning controllers. In: Intelligent Virtual Agents - 15th International Conference, IVA, 2015, Delft, The Netherlands, August 26-28, 2015, Proceedings. (2015) 119–123

[6] dos Santos, L.A.: Artificial intelligence. https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/recurrent_neural_networks.html Accessed: 2017-04-20.

[7] Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10**(2) (2002) 99–127

[8] Ihar Kalasouski, E.C.: Box2dx. https://code.google.com/archive/p/box2dx/

[9] Green, C.: Sharpneat. http://sharpneat.sourceforge.net (2003-2017)

[10] Autodesk, Inc.: Maya. https://www.autodesk.com/education/free-software/maya

[11] 3dregenerator: Agent smith model. https://free3d.com/3d-model/agent-smith-12422.html

[12] SD, V.: Ac cobra mkii model. https://www.turbosquid.com/FullPreview/Index.cfm/ID/1017130

[13] Maurhofer, C.: Crowd Simulation - A Python Framework for the Simulation of Human Actors in Miarmy (2016)

[14] Torrey, L.: Crowd simulation via multi-agent reinforcement learning (2010)

[15] Papis, B.: dotrl: A platform for rapid reinforcement learning methods development and validation