The Penguin AI

Jakob Bertlin, Media Technology Student at Linkoping University Jakbe457@student.liu.se
Simon Corell, Media Technology Student at Linkping University, Simco664@student.liu.se

***Abstract*—This report describes how a genetic algorithm with a neural network can be implemented and used to learn a AI-bot to master a 2D platform game. The focus of the report is to inform the reader how a genetic algorithm combined with a neural network is implemented and how it can be applied to solve a rather complicated problem.**

I. Introduction

Artificial intelligence is something that almost everyone have heard of by now. It is considered being our time next big thing and many companies are trying to use it in their business. Genetic algorithms and neural networks are both artificial intelligence methods that can be used to solve many different problems. Genetic algorithm is based on Darwin's theory about survival of the fittest and natural evolution. Neural network tries to mimic how our human brains work and interact. After reading this report the reader hopefully will understand how powerful these two methods combined really are, and how they can be used to learn a AI-bot master a rather complicated 2D platform game called Super Tux 2.

II. Super Tux 2

Super Tux 2 is a Linux based 2D platform game, the source code of the game is open to everyone and can be found on Github [1]. The primarily goal is to reach the goal at the far right end of the level. The game is similar to the old 2D Super Mario where the player always tries to get further to the right until it reaches the end. The player has three different controller inputs; go right, go left and jump. The player will face static obstacles such as wall and holes. Other more dynamic obstacles the player will face are enemies. The player dies if it touches the enemy, but the player can also jump on it to kill the enemy. The game is deterministic in that sense that the enemies will always spawn at the same time and move with the same velocity and direction. This also applies to all the static obstacles, the level will always look the same. The game is written in C++ and our AI is implemented in the source code. The AI has been written without any third party library. In figure 1 you can see a typical frame of the game.

III. Genetic Algorithm and Neural Network

A. *The Genetic Algorithm*

A genetic algorithm is inspired by the process of natural selection and belongs to the larger class of evolutionary algorithms. Genetic algorithms are commonly used to solve tasks like optimization and search problems but can also be applied to varies numbers of problems, such as mastering a computer game. The main components of a genetic algorithm are initial population, reproduction, mutation, determination of fitness and selection. Each one of these components depends on each other and each has to be carefully selected with respect to the desired result.

The basic idea of the genetic algorithm is to always create new generations of objects with desired behaviour and this way mimic the natural selection. At every generation the most desired objects are chosen for the next generation until the satisfied behavior is reached. How similar the new generation of objects should look like the previous best is chosen by the mutation. The fitness value is how the desired behaviour should be estimated and the selection consists of how many and how the selecting of the objects should look like. The reproduction is how the new generation should look like with respect to the selected ones. [2]

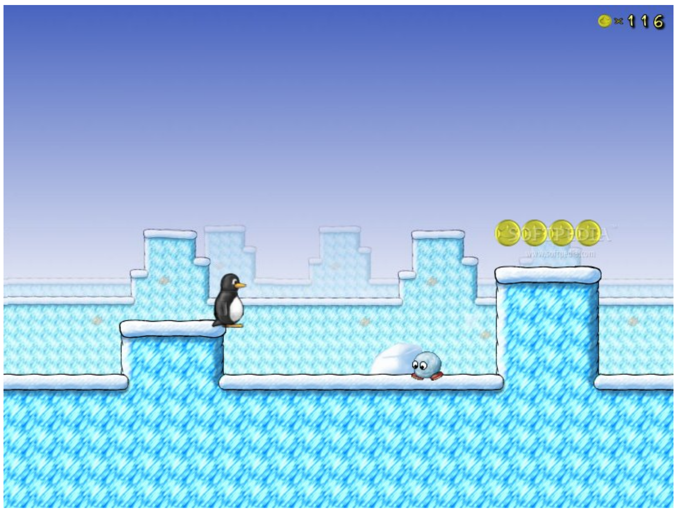

Fig. 1. A typical view of how the game Super Tux 2 looks like, where the player is the penguin.

B. *The Neural Network*

Neural networks are learning systems inspired by the biological neural networks in human brains. It is itself not an algorithm, but rather a framework to many different algorithms for example in machine learning. Its purpose is to learn to perform tasks by considering examples without hard coded rules. It is commonly used in problems like image recognition. A neural network is based on connected nodes called neurons, these should model the neurons in a human brain. Each node have a weight that changes as the learning proceeds. The

Advisor: Dipl.–Ing. Pierrangelo Dell'Acqua, weight represents how strong of an impact the neuron should have. Each connection between the neurons should model a synapse in a human brain. The neurons belongs to different layers. The layers are called the input layer, the hidden layers and the output layer. This can be seen in figure 2 below.

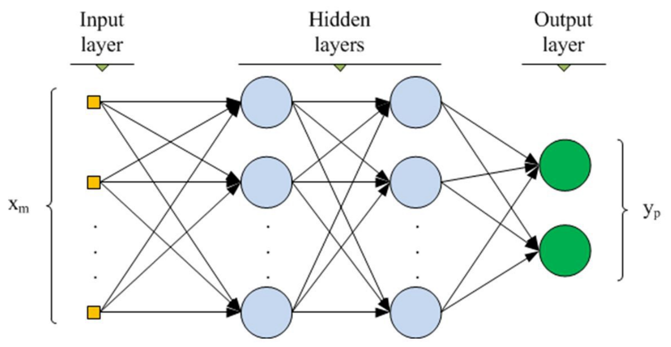

Fig. 2. An example two hidden layer neural network.

The basic idea of a neural network is to have inputs to the input layer as numbers and letting the input layer transmitting the inputs through the network. Every neuron outputs a number which is based on a non-linear computation of the sum of its inputs. Every neuron later transmits this number to all the neurons in the next layer. This continuous until the inputs reaches the output layer which outputs the result. [3]

IV. Method

A. Implementation of the genetic algorithm

How the flow block for the genetic algorithm used in this method looks like can be seen in Fig. 3.

As you can see in the figure, the first thing that had to be chosen is the initial population. The initial population that was chosen was a neural network with random weights between -1.0 and 1.0. This because it is impossible to know exactly which values the weights should have before the learning process has even started. Since the genetic algorithm is based on survival of the fittest the reproduction consisted of adding the four currently best networks and computing a weighted mean of these four. A new generation is computed continuously every time the current network has failed, letting the four currently best networks create a new offspring for every try. The reproduction also take more in consideration for the first best network and less consideration for the fourth best network. This was achieved through multiplication of the best network with 0.4, the second network with 0.25, the third network with 0.20 and the fourth network with 0.15. A mutation is then applied to the newly spawned network for how much it should resemble it's "parents". A higher mutation aims for a more free movement from it's parents which allows for discovering new behaviour, but it might also diverge too much from a good path that it's parents achieved. This is particular needed if the AI have learn a certain behaviour that is wrong or bad. The mutation used in this genetic algorithm starts at 1 percent and increases over time up to 50 percent if no progress has been made. It will reset the mutation back to 1

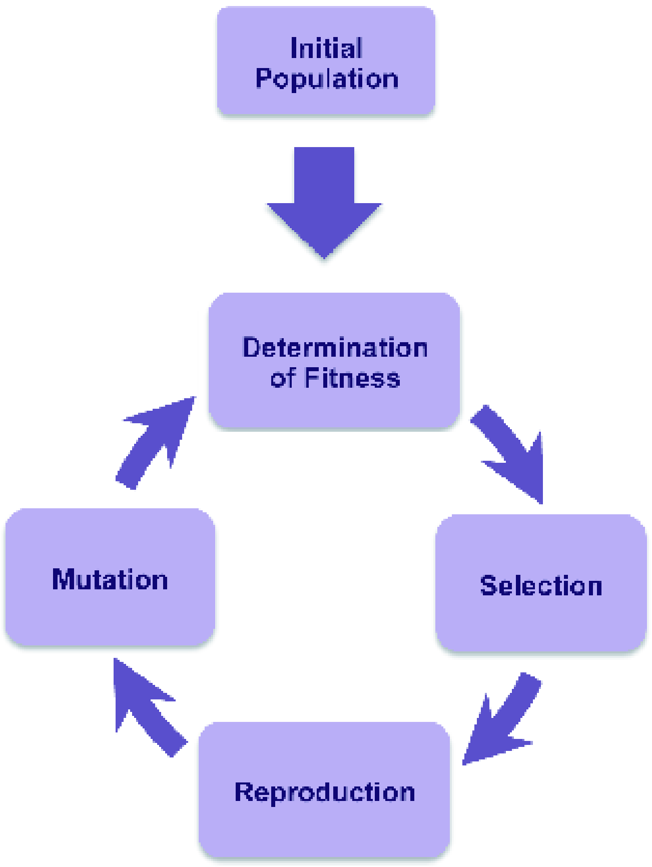

Fig. 3. The block scheme over the genetic algorithm.

percent if it still has made no progress at 50 percent mutation. Determination of fitness is how far the AI have reached in the level. That is, the x-coordinate the AI had when it died. So the further the AI have reached in the game, the higher will the fitness value be. So the selection consists of choosing the four networks with the best fitness values.

B. Implementation of the neural network

How the neural network used in this method looks like can be seen in the figure below. The neural network used in this method consists of three different type of layers. The input layer, the hidden layers and the output layer. The input layer takes the inputs and traverses the inputs through the layer. The hidden layers traverses the outputs from the input layer through its layers. The output layer traverses the output from the hidden layer and outputs the result of the network. There are four inputs to the input layer. The first two inputs are the distance in the x direction to the two closest obstacles of either a wall or/and enemy. The third input is the elevation in front of the AI-bot. The fourth last input is the elevation directly under the AI-bot. These inputs are computed from variables within the source code. This was done with ray traces which calculates the distance traveled before hitting an obstacle. The inputs can be seen in figure 4. There are only 2 outputs which are jump and walk right. If the value of the first output is above 0.0 the player will jump, otherwise it will not jump. If the second output is above 0.0 the player will move to the right and if it is below it will stand in place.

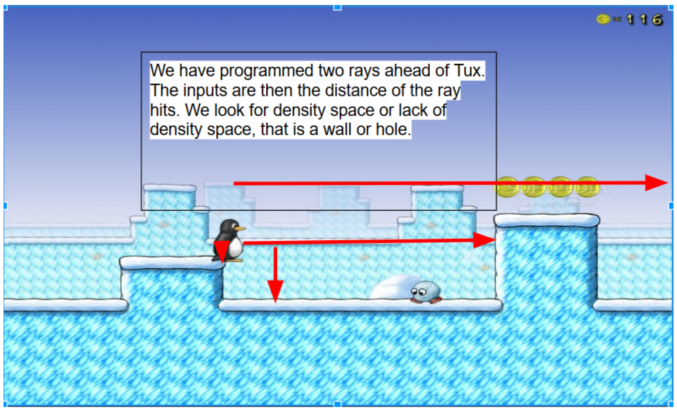
116
We have programmed two rays ahead of Tux. The inputs are then the distance of the ray hits. We look for density space or lack of density space, that is a wall or hole.

Fig. 4. The inputs to the Neural Network

V. Results

After training the AI-bot for 30 minutes it had learned to successfully complete the first level of the game. After this amount of time the AI-bot almost completes the level on every try. It jumps at the right time whenever a enemy is within a certain distance. It jumps up to higher grounds whenever it is needed to get further into the level. This is because the inputs will always generate weights that will choose the best output possible, generating a behaviour that is satisfied enough to be able to complete the level. When testing the AI-bot on other levels, it is shown that its behaviour is intelligent, in that sense that it is jumping over obstacles and enemies. But whenever a new phenomena is presented in the level, for example a new kind of enemy, the AI-bot have difficulties to proceed. But with some training, the AI-bot will probably complete the level.

VI. Conclusion and future work

We are indeed satisfied with choosing a genetic algorithm with a neural network as method for our AI-bot. The method have shown to be a strong one for learning a bot to behave in a intelligent way. One thing that we want to address is the importance of choosing smart inputs and outputs to the network. We tried many different strategy with different inputs to the network which gave far worse results. In the start we had left, right and jump as outputs. With these outputs the AI-bot had a hard time learning a satisfied behaviour. The AI-bot was for example standing still for a long time meaning it was choosing between left and right. Because of this we choose to only have right and jump as outputs. This also because the AI-bot should always proceed to the right, leaving going left just unnecessary. In the beginning we had the distances to the two closest enemies in the world as inputs to the network together with the distance to the closest obstacle. This was working rather well but we figured that we wanted to be able to give the network more generalized inputs. As a result to this we created the inputs that we have presented in this report.

As for the results itself, we are pleased with that to. Our primary goal for the AI-bot was to complete one level in the game and it have learned just that. The AI-bot can be applied to other levels as well but not with the same results. Further work with the AI-bot is to teach it to master the whole game. This can be done with some improvements. For example we could create a genetic algorithm which covers the whole game. This meaning that we save the best neural network for each level and then for every new generation reproduce a new neural network. After training the network it could in the end create a neural network that can complete every level in the game. Other features that could be added to the AI-bot are for example, coin balance as fitness value and maybe add some kind of punishment when doing errors.

References

[1] https://github.com/SuperTux/supertux
[2] Kramer, Oliver. *Genetic Algorithm Essentials*. Springer, Cham., New York, USA, first edition, 2017.
[3] Grgoire Montavon, Genevive B. Orr, Klaus-Robert Mller (eds.).*Neural networks, tricks of the trade*. Springer. New York, USA, second edition, 2012.