

Training an agent using neural networks

Perttu Jääskeläinen & Evan Saboo



# Initial study

In this chapter, a short background will be given about the field which this project will dive into. This chapter will also define functional and non-functional requirements for this project.

## Background

This project focuses on Artificial Neural Network by training the neural network to be able to find points in a game map as the environment. Artificial neural network (ANN) is algorithms that find relationships in a dataset similar to how humans think. ANN are based on the same principle as a human brain, as people learn through their surroundings and then save the information. In order for an artificial neural network to find relationships, it must first be trained. They learn by training them with examples where the correct result is known.

The method used to train the model is call NEAT. NEAT, or Neuro-Evolution of Augmenting Topologies, is a method for evolving artificial neural networks with a genetic algorithm, introduced by Kenneth O'Stanley and Risto Miikkulainen. The method is to start with a simple neural network and then allow it to become more complex with each new generation.

## Goal

The goal of this project is to train a player in a game map to avoid obstacles and find points which are randomly place in the map. The functional and non-functional requirements for the project are presented below.

* + 1. **Functional requirements**
* Use the NEAT method to train the network to complete different game maps.
* The game and the NEAT method should be implemented in python.
* The player can reach the goal by searching for the reward and avoid all obstacles in the given game environment.
* The game should be designed and implemented without using any external game library.

## Non-functional requirements

* The player should be able to handle with several types of maps.
* The player should be able to handle from small to large game maps.
* It should be able to reach the reward with limited number of moves.

# Design

The design of the map in which the neural net (NN) navigates should contain obstacles, free spaces, a reward and a current position of the AI/NN. When reaching the reward, a new reward should automatically spawn in a randomized position on the map. The NN should be trained on various maps, where the sizes and shapes of the obstacles vary, so that it can learn to recognize patterns instead of just a sequence of operations for moving around. The design is shown in figure 2.1

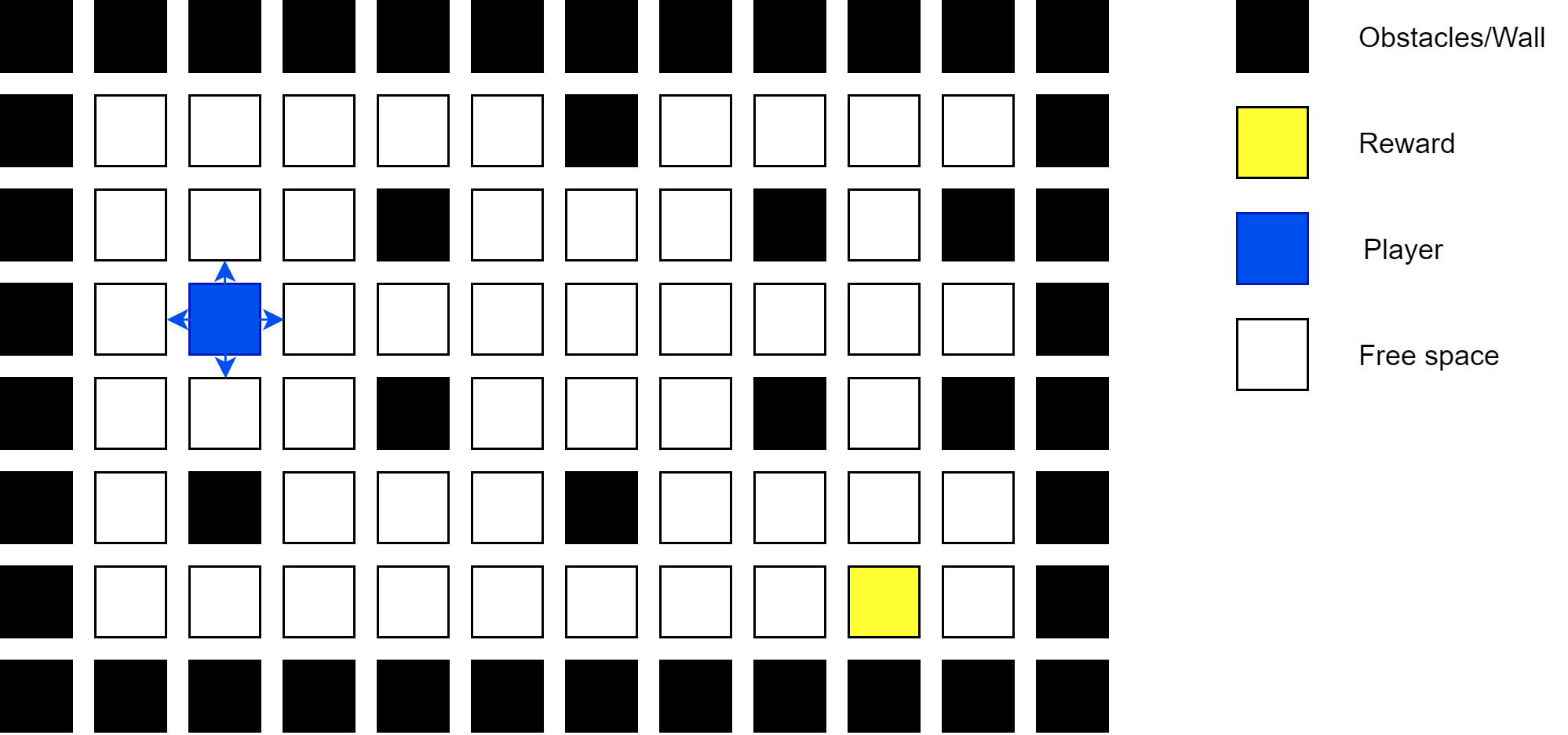


Figure 2.1: The map layout. Split into obstacles, free spaces, the players position and the current position of the reward. When reaching the reward, a new one should spawn at a random location.

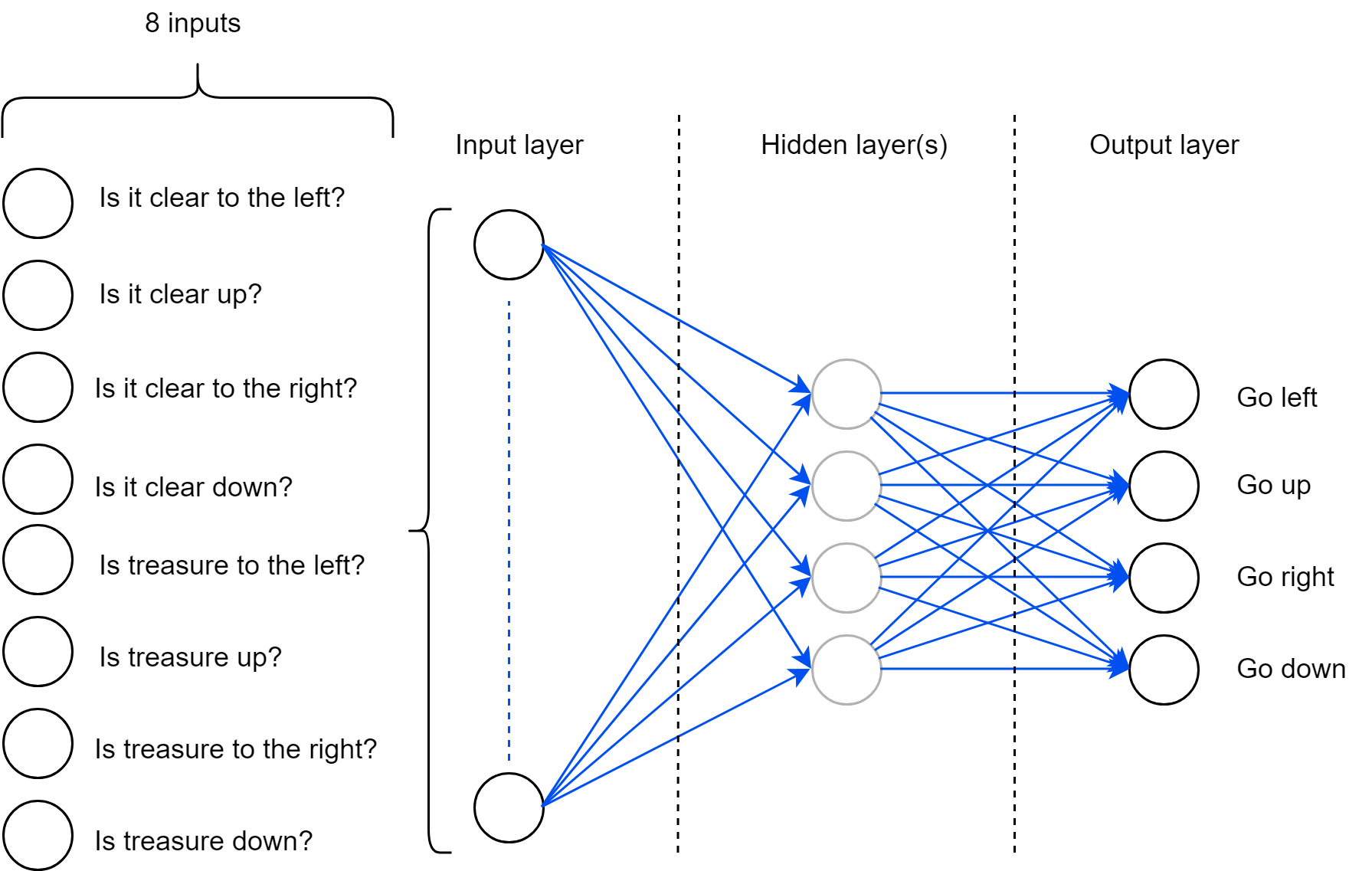


Figure 2.2: The design of the neural network.

The design of the NN should be in a way that it can recognize its surroundings while still being aware of the position of the goal. It should also be able to adapt to situations where the map size is undefined, only going by the direction in which the goal is and which of its surrounding positions are clear to traverse.

To do this, it is enough to define the positions in the following way: 4 indexes to define if left, up, right or down is clear, and another 4 to describe if the reward/goal is to the left, up, right or down. This way, all directions can be represented (up-left as north west, up-right as north east etc.). This is shown in figure 2.2.

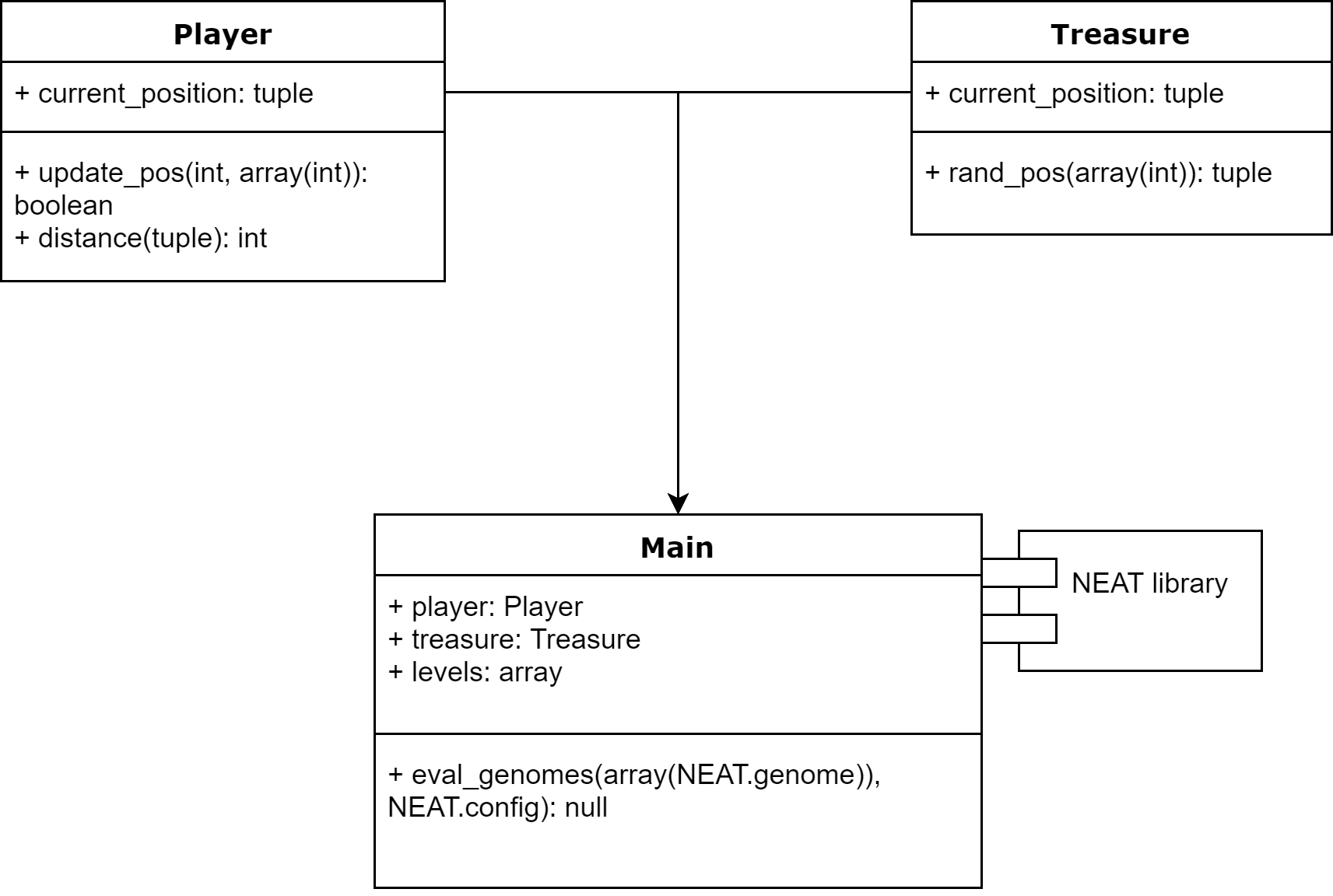


Figure 2.3: The code design, where the function ‘eval\_genoms(...)’ is run until the desired fitness is reached, defined inside the NEAT.config

The NN should then randomize its next movements depending on the data received from calculating the surroundings of the player and difference to the position of the treasure/goal. When the NN decides on an action, the next state should be calculated and the NN should either be rewarded for its action, stay the same (if moving in a direction that does not change its distance to the goal), or terminate if walking into a wall.

The design of the code should be in a way such that the position of the player, treasure (goal) and map layout is available at all times. If wanted, it should also be possible to navigate between maps to variate training data for the NN to practice on. The distance between the player and a treasure is calculated using the function distance() in the player class. The function uses pythagoras theorem to calculate the distance. The distance is then used to check if the player is closer its previous best distance and reward the NN if that is true.

# Implementation

To implement the neural network, we used the NEAT Python library[[1]](#footnote-0), created by CodeReclaimers. Using the library, we created rewards for the neural network (NN) when it reached a new, closest distance to the current goal, and when reaching the goal itself. This was done in increments, where the NN got the following rewards, where it continued running until the initially set goal was reached:

0 - when moving away from the goal

1 - when reaching a new, closest distance to goal (reset when goal is reached)

10 - when the goal is reached

This worked out the best, where we tried various reward values and tested them out. By rewarding the net when moving toward the goal, even if it wasn’t a new record, it could navigate around obstacles and still receive some sort of reward for doing so.

The map had to be designed in a way where the NN could easily identify and differentiate between free spaces, obstacles and the goal. We used the following values for the objects:

0 - Free space where the AI can move

1 - An obstacle (wall). When the AI moves into an obstacle, it instantly dies

2 - The initial starting- and current position

3 - The goal to be reached

One of the game maps is shown in figure 3.1 as an array and the corresponding visual interface is shown in figure 3.2.

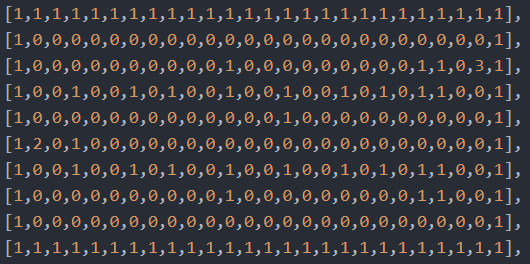


Figure 3.1: One of the game maps presented in a array. Number 0 is free space, 1 is an obstacle, 2 is the player and 3 is the treasure.

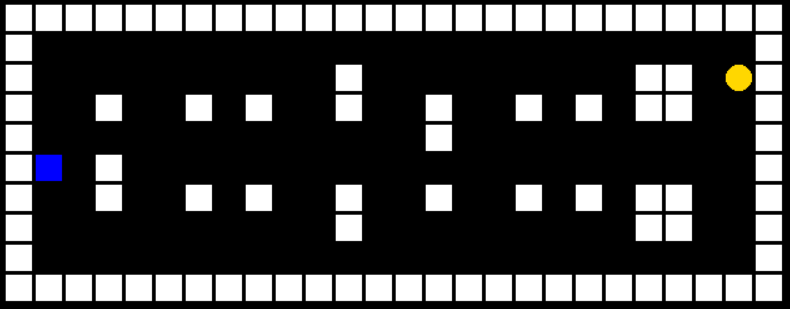


Figure 3.2: One of the game maps where the NN learned in.

# Testing

Four tests were performed to check if the player could handle a typical map with obstacles. Every test consisted of a predesigned map which the player would be placed on. In this map, the players task was to find a total of 100 goals which were randomly spawned. The player had to avoid every obstacle in the map when finding the goals, overwise the player would immediately fail the given test.

The first map shown in figure 4.1 does not have any obstacles and was the easiest succeed on. The second map shown in figure 4.2 introduced some 1x1 obstacles which the player have to avoid in order to stay alive. The third map shown in figure 4.3 introduces more 1x1 size obstacles which would make it a bit more difficult to avoid. The final map shown in figure 4.4 is the hardest test to succeed on because it introduces 2x1 and 2x2 size obstacles.

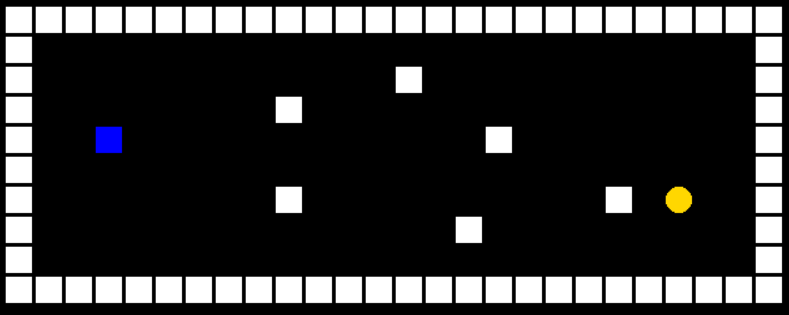


Figure 4.1: Map without obstacles Figure 4.2: Map with 1x1 obstacles

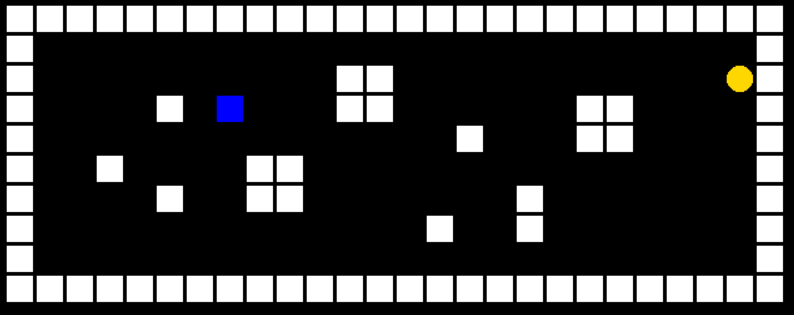
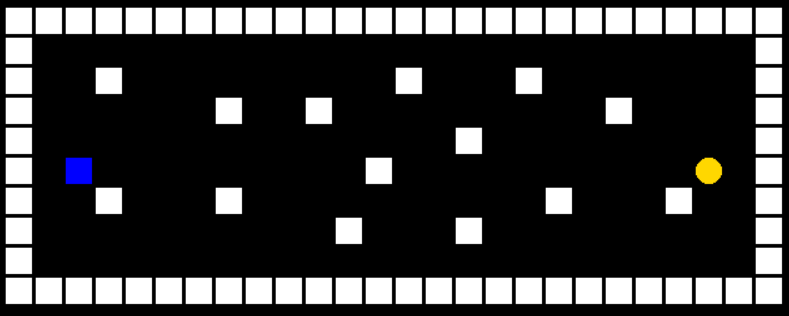


Figure 4.3: Map more 1x1 obstacles Figure 4.4: Map with 2x2. 2x1 and 1x1

obstacles

After all four tests were completed, acceptance testing was performed which checked if all functional and nonfunctional requirements were met.

# Delivery

To use the system, the user has to input a map with the following format:

0 - Free space that the player (AI) can move to

1 - A wall, where the player will “die” on collision

2 - Starting position of the AI

3 - The goal position

The map is represented as an multidimensional array, where each index represents a row on the map (as an array), where each index in the second array represents the columns on the row.

Each map has to have exactly one starting position (2) and one goal (3). The AI will then automatically place out new, randomly generated goals when the first goal is reached while the players position is updated as the AI maps its input to output and navigates around the map.

The design is up to the user, there are no limitations or designs that are not allowed. However, it is advised that the maps are designed such that the neural net learns from wall designs in increments, such as starting with ‘2x1’ walls and moving on to ‘3x1’ or ‘2x2’ structures. Multiple structures may be combined freely where smaller structures are easily recognized by the AI while larger ones become increasingly difficult depending on previous learning.

The AI will practice on the given map by navigating to the goal while avoiding collision with walls. Depending on the set goal state (how much fitness the neural net needs to achieve by reaching goals), the AI will keep running until it ‘figures out’ the maps wall structures and patterns.

After finishing practice by reaching the goal fitness, the same neural net can be used on a map with a similar wall structure and will be able to complete it within reasonable time.

The code can be accessed by going to this link (<https://gits-15.sys.kth.se/saboo/AI-Projekt>). The game and the neural network executes by running the file main.py.

**References**

Artificial Neural Networks Technology. *University of Toronto* . Retrieved from <http://www.psych.utoronto.ca/users/reingold/courses/ai/cache/neural2.html> (2019-01-09)

Stanley, Kenneth O., and Risto Miikkulainen. "Evolving neural networks through augmenting topologies." Evolutionary computation 10.2 (2002): 99-127.

Artificial Neural Networks Technology. *CodeReclaimers*. Retrieved from <https://neat-python.readthedocs.io/en/latest/> (2019-01-09)

1. <https://neat-python.readthedocs.io/en/latest/neat_overview.html> [↑](#footnote-ref-0)