UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

# Learning to solve Sliding Puzzles using Reinforcement Learning

Umr Barends
18199313

Report submitted in partial fulfillment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at Stellenbosch University.

Supervisor: JC Schoeman

October 12, 2020

# Acknowledgements

I would like to thank my dog, Muffin. I also would like to thank the inventor of the incubator; without him/her, I would not be here. Finally, I would like to thank Dr Herman Kamper for this amazing report template.

UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

# Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   *Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.*

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   *I agree that plagiarism is a punishable offence because it constitutes theft.*

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   *I also understand that direct translations are plagiarism.*

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   *Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism*

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   *I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.*

| | |
|---|---|
| Studentenommer / *18199313* | Handtekening: [illegible] |
| Initials and surname / *U.Barends* | Datum / *October 12, 2020* |

# Abstract

## English

The English abstract.

# Contents

# List of Figures

# List of Tables

# Nomenclature

## Variables and functions

| | |
|---|---|
| $p(x)$ | Probability density function with respect to variable $x$. |

## Acronyms and abbreviations

| | |
|---|---|
| RL | Reinforcement Learning |
| SARSA | State-Action-Reward-State=Action |
| Q-learning | Q value learning |
| MDP | Markov Decision Process |

# Chapter 1

# Introduction

## 1.1. Background

In the modern world robots are becoming more and more part of our daily lives and society. For example the Xiaomi house cleaning robot. Generally in robotics a manipulator (eg. an arm) is used to manipulate an object in the environment. It is easier to first simulate the robots behavior in a more simple environment which is why we will solve a similar and smaller problem as a step to a complete solution.

## 1.2. Problem Statement

In this project we solve sliding puzzles using reinforcement learning, where the same algorithm can then later be applied to the robotics problem of moving in an environment.

## 1.3. Project Objectives

- Solve a 2x2 sliding puzzle using Reinforcement learning
- Solve a 3x3 puzzle using Reinforcement learning
- Puzzles must solve in a reasonable amount of time

## 1.4. Demonstration video

## 1.5. Scope

Although there are many methods of solving a problem in RL, in this project we only look at two methods. These are SARSA and Q-learning. Typically for problems with large state spaces neural networks are used in conjunction with the RL methods, to save computational time. However for this project we used a certain method to overcome the state space limitation which will be discussed later.

## 1.6. Report Overview

# Chapter 2

# Literature Review

## 2.1. Introduction

Reinforcement learning is a method of learning what to do by linking states to actions with a numerical reward incurred for every state-action pair. The final goal of RL is to find a path of states and actions with the maximum amount of reward. [3]

## 2.2. Required knowledge

### 2.2.1. Markov Decision Processes (MDP's)

To model reinforcement learning problems we use dynamic systems theory, specifically using incompletely-known MDP's (Markov decision processes). Most of RL problems can be described using MDP's. In Mathematics a MDP is a stochastic, discrete time control process. What that means is that the process is essentially partially controlled and partially random. MDP's depend mainly on a few variables which are, states, actions, state transition probability ,reward and a discount factor [3]. These variables can be denoted in a 5-tuple as:

$$(S, A, P^a_{ss'}, R^a_s, \gamma)$$

where

- S is a finite set of states
- A is a finite set of actions
- $P^a_{ss'}$ is a matrix of probabilities with $P^a_{ss'} = P(S_{t+1} = s'|S_t = s, A_t = a)$
- $R^a_s$ is the immediate reward after transitioning from state s to s' using action a. Where $R^a_s = E[R_{t+1}|S_t = s, A_t = a]$
- $\gamma \in$ [0,1] is the discount factor applied to the reward

For a state to be Markov it needs to satisfy the following condition:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t]$$

This means that the current state is required to contain all the information of the previous states. For a MDP all states must be Markov.

We now define the definition which is the total discounted reward from time-step t, named the return $G_t$ where:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... \tag{2.1}$$
$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

The discount $\gamma \in$ [0,1] determines the present value of future rewards. For $\gamma = 0$ the return $G_t$ only depends on the current reward that can be obtained in the next step $R_{t+1}$. Which can be said to be "short-sighted". While for $\gamma = 1$ the return depends on all the rewards that are projected to be obtained until the process terminates. This can be said to be "far-sighted". The larger $\gamma$ is the more the rewards of later steps closer to the terminating state affects the return. [3] What we can also note is that using a discount factor makes the return finite because $R_{t+k+1}$ is finite and:

$$\sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

which is finite for $\gamma \in$ [0,1].

We now define another important required definition, namely the state value function v(s) where:

$$v(s) = E[G_t|S_t = s]$$

Which in word mean that the value state function v(s) is the expected return given that the agent is in state s at time-step t. We now decompose v(s) so that it becomes a recursive function as follows:

$$v(s) = E[G_t|S_t = s] \tag{2.3}$$
$$= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s] \tag{2.4}$$
$$= E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...)|S_t = s] \tag{2.5}$$
$$= E[R_{t+1} + \gamma G_{t+1}|S_t = s] \tag{2.6}$$
$$= E[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \tag{2.7}$$

This means that v(s) only depends on $R_{t+1}$ the reward it can incur in the next step as well as $v(S_{t+1})$ the discounted state function at time t+1. What we have just defined is known as the Bellman Equation. [3]

The probability an agent takes a certain of action given it is in a given state is defined as a policy $\pi(a|s)$. It is defined as:

$$\pi(a|s) = P[A_t = a, S_t = s]$$

Then we can now define the state value function following policy $\pi$ as:

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

Additionally we define the action-value function also known as a q-function as:

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$$

The q-function is defined as the expected return taking action a from state s thereafter following policy $\pi$. What can be noted is that v(s) is a prediction of what the value for being in a certain state is. While q(s,a) is the value for being in a certain state and taking a action, which has to do with control compared to v(s) which has to do with planning.

Once again we can decompose the state value and also action value functions to be in the form of the Bellman Equation in equation 2.7. This results in the Bellman *Expectation* Equations:

$$\begin{aligned} v_\pi(s) &= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] && (2.8) \\ &= \sum_{a' \in A} \pi(a|s)(R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v_\pi(s')) && (2.9) \end{aligned}$$

$$\begin{aligned} q_\pi(s, a) &= E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] && (2.10) \\ &= R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a' \in A} \pi(a'|s') q_\pi(s', a') && (2.11) \end{aligned}$$

The Bellman Expectation Equations can also then be put into matrix form from equation 2.9 for fast vector calculation as:

$$v_\pi = R^\pi + \gamma P^\pi v_\pi \tag{2.12}$$

Which has the solution:

$$v_\pi = (1 - \gamma P^\pi)^{-1} R^\pi \tag{2.13}$$

Equation 2.13 is a fast and concise formula which is easily implemented in code, but has the drawback of only working for small state spaces. This is due to the fact that the calculation required the entire transition matrix $P^\pi$ to be loaded into memory at every calculation step.

We now define what is known as the optimal policy, which is essentially a set of state

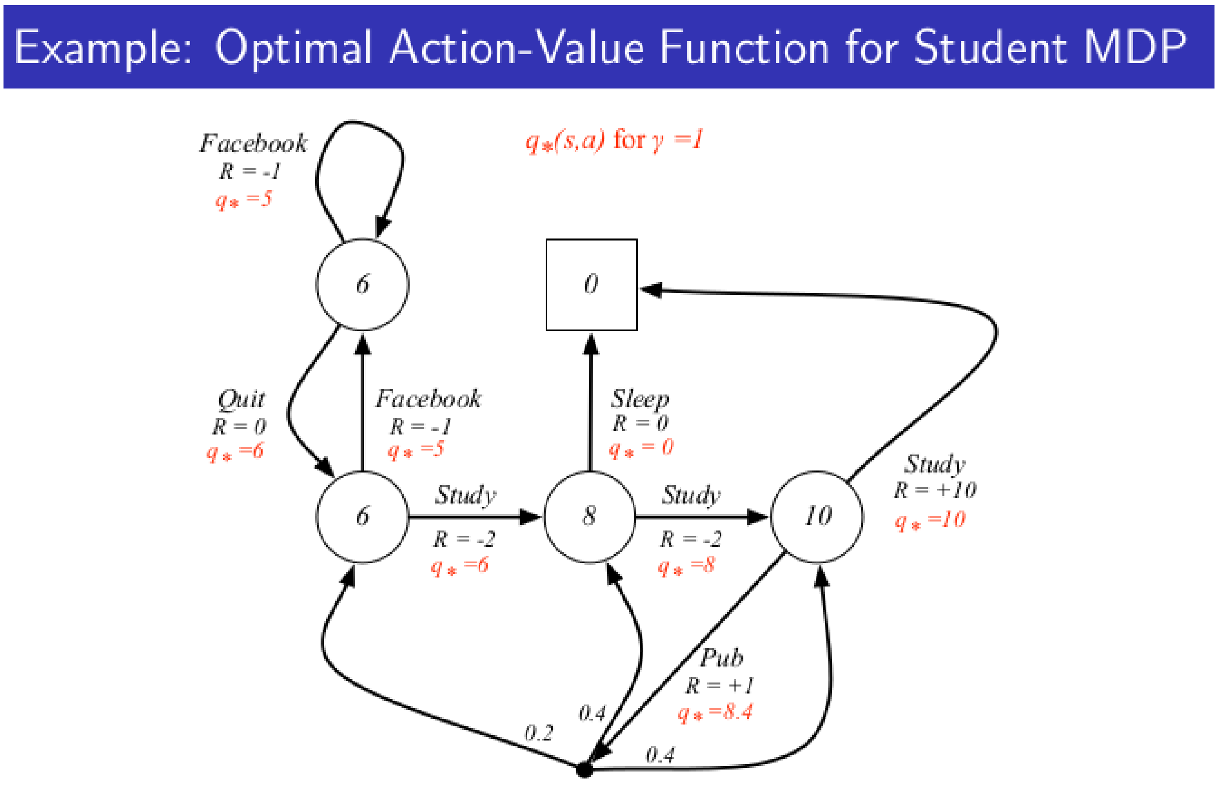


**Figure 2.1:** Graph of an example MDP showing how q values work [1]

and action pairs which lead to the maximum $q_\pi(s,a)$ as:

$$\pi_*(a|s) = \begin{cases} 1, & \text{if a} = \underset{a \in A}{\arg\max}\, q_*(s,a) \\ 0, & \text{otherwise} \end{cases} \tag{2.14}$$

There always exists an optimal deterministic policy for any MDP [3]. We only need to know $q_*(s,a)$ to know the optimal policy. This is important because the optimal policy describes to us how the agent must move to maximize its rewards, which is precisely the goal of RL. Figure 2.1 shows an example MDP displaying how $q_*(s,a)$ works. The red path lines indicate the optimal path/policy in Figure 2.1.

We now finally define what is know as the Bellman *Optimality* Equations which is a specific form of equation 2.7 and 2.9 which are:

$$v_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \tag{2.15}$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s',a') \tag{2.16}$$

What equation 2.15 represents is the maximum reward that can be obtained in state s. While equation 2.16 represents the maximum reward that can be obtained by taking action a from state s. The difference between the two is that equation 2.15 is just a passive prediction of what value (according to reward that can be obtained) a state represents. While equation 2.16 is a calculation of the value of being in a state based on after a action 'a' is selected.

### 2.2.2. Model free control

## 2.3. Solving sliding puzzle using value iteration

In [4] they use what is called value iteration

- objectives
- method
- results and what they didn't do that if they did would have addressed your problem
- remaining challenges

# Chapter 3

# System Design

insert software diagram here

## 3.1. Thing 1

To solve the puzzle I could have used monte carlo learning, .... I chose SARSA and Q-learning because ...

## 3.2. Thing 2

## 3.3. Methods

- Measure reward-episodes graphs

# Chapter 4

# Detailed Design

## 4.1. Thing 1

Show functional block in detail of thing 1

## 4.2. Thing 2

Show functional block in detail of thing 2

# Chapter 5

# Results

## 5.1. SARSA figs

## 5.2. Q-learning figs

## 5.3. Discussion

# Chapter 6

# Reinforcement Learning Theory

Reinforcement learning is a method of learning what to do by linking states to actions with a numerical reward incurred for every state-action pair. The final goal of RL is to find a path of states and actions with the maximum amount of reward. [3]

To model reinforcement learning problems we use dynamic systems theory, specifically using incompletely-known MDP's (Markov decision processes). Most of RL problems can be described using MDP's. In Mathematics a MDP is a stochastic, discrete time control process. What that means is that the process is essentially partially controlled and partially random. MDP's depend mainly on a few variables which are, states, actions, state transition probability ,reward and a discount factor. These variables can be denoted in a 5-tuple as:

$$(S, A, P^a_{ss'}, R^a_s, \gamma)$$

where

- S is a finite set of states
- A is a finite set of actions
- $P^a_{ss'}$ is a matrix of probabilities with $P^a_{ss'} = P(S_{t+1} = s'|S_t = s, A_t = a)$
- $R^a_s$ is the immediate reward after transitioning from state s to s' using action a. Where $R^a_s = E[R_{t+1}|S_t = s, A_t = a]$
- $\gamma \in$ [0,1] is the discount factor applied to the reward

For a state to be Markov it needs to satisfy the following condition:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t]$$

This means that the current state is required to contain all the information of the previous states. For a MDP all states must be Markov.

We now define the definition which is the total discounted reward from time-step t, named the return $G_t$ where:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + ... \qquad (6.1)\\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad (6.2) \end{aligned}$$

The discount $\gamma \in [0,1]$ determines the present value of future rewards. For $\gamma = 0$ the return $G_t$ only depends on the current reward that can be obtained in the next step $R_{t+1}$. Which can be said to be "short-sighted". While for $\gamma = 1$ the return depends on all the rewards that are projected to be obtained until the process terminates. This can be said to be "far-sighted". The larger $\gamma$ is the more the rewards of later steps closer to the terminating state affects the return.

We now define another important required definition, namely the state value function v(s) where:

$$v(s) = E[G_t|S_t = s]$$

We can also decompose v(s) so that it becomes a recursive function as follows:

$$\begin{aligned} v(s) &= E[G_t|S_t = s] \qquad (6.3)\\ &= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s] \qquad (6.4)\\ &= E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...)|S_t = s] \qquad (6.5)\\ &= E[R_{t+1} + \gamma G_{t+1}|S_t = s] \qquad (6.6)\\ &= E[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \qquad (6.7) \end{aligned}$$

# Chapter 7

# Solvability of a NxN sliding puzzle

## 7.1. General puzzle description

Let us assume that we have an NxN puzzle, then we have NxN number of blocks. We can represent the puzzle as an NxN array, then we stack the array into a one dimensional array of 1 x (N*N). For example see the 4x4 puzzle in Figure 7.1 we have a 1 x 16 array as: Array = (12,7,8,13,4,9,2,11,3,6,15,14,5,1,10). Before we describe the conditions for a sliding puzzle to be solvable, we first define the term "inversion". Assuming the the first index of the 1xN 2 array starts at the left top corner (valued 12) in Figure 7.1, and that it runs from [0,(N*N)-1]. Then an inversion occurs when Array[index] ¿ Array[index+1] where index is an arbitrary integer between 0 and N*N-1. Hence in Figure 7.1 we have a total: sum of inversions(Array) = 11 + 6 + 6 + 8 + 3 + 5 + 1 + 5 + 1 + 2 + 4 + 3 + 1 + 0 = 56.

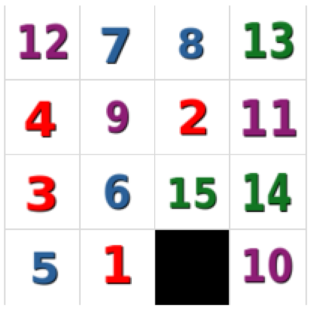


**Figure 7.1:** Example of a sliding puzzle

## 7.2. Conditions for solvability

Even and odd sized boards are analysed separately (where size = N).

For odd sized boards where N is odd we have the puzzle only being solvable if and only if the boards has an even number of inversions. The proof for this can be deduced by looking at Figure 2 and noting that for every switch of the blank block we have an even change in the sum of inversions of the board. [2]

For even sized boards where N is even we have the board solvable if and only if the

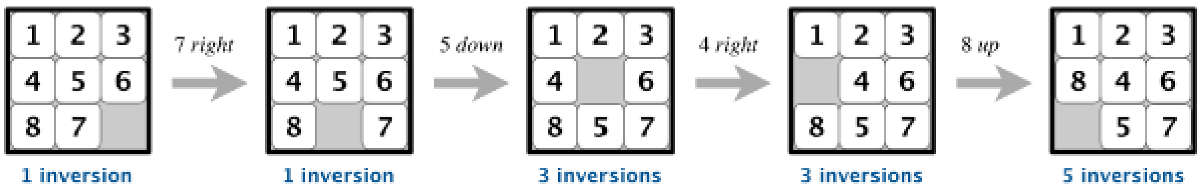


**Figure 7.2:** Odd boards with change in blank piece only having even inversion change [2]

number of inversions plus the row of the blank square is odd. This is illustrated in Figure 3.

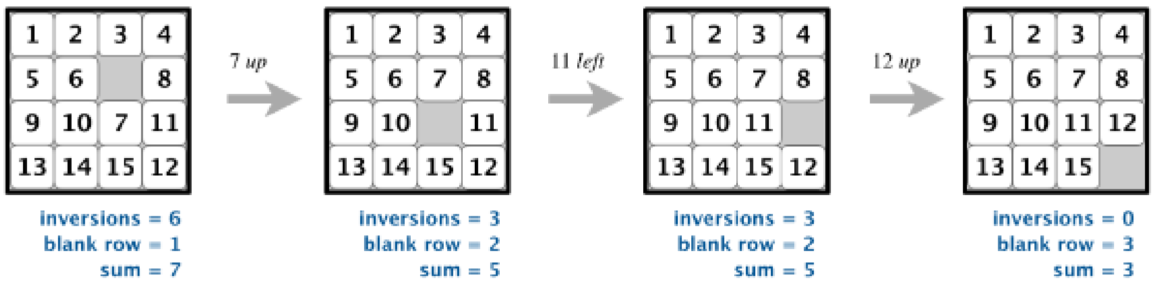


**Figure 7.3:** Even board solvability [2]

Half of all puzzle configurations are unsolvable. [5] This means that we only have N! / 2 configurations that are solvable for an NxN board. This was proven using parity in the paper in [5]. Sliding puzzles can be solved relatively quickly with today's processing of computers for puzzles for example an 5x5 puzzle was solved in 205 tile moves in 2016. [6]

The issue more so lies in finding the shortest path to solving a puzzle. This specific problem of solving with the least amount of tile moves of a sliding puzzle has been defined as NP (non-deterministic polynomial-time) hard. NP hardness is are problems that are as least as hard as NP. Where in computational complexity theory NP (non-deterministic polynomial-time) is a has a solution with a proof variable to be in polynomial time by a deterministic Turing Machine. A Turing machine is a mathematical model defining an abstract machine which manipulates symbols according to a set of rules. [7]

In simpler terms a problem is NP if it can be solved within a time that is a polynomial function of the input. For instance if we define the time to solve a problem as 'T' and the input data as 'D'. Then as long as T = polynomial function (D) then a problem is NP.

# Chapter 8

# Summary and Conclusion

## 8.1. Summary

Objective 1 ...

Objective 2 ...

Objective 3 ...

So what? Relates back to problem and background sections in introductions.

## 8.2. Future work and reccomendations

# Bibliography

[1] D. Silver, “Ucl course on rl,” 2015. [Online]. Available: https://www.davidsilver.uk/teaching/

[2] P. U. F. of Computer Science, “8puzzle assignment.” [Online]. Available: https://www.cs.princeton.edu/courses/archive/spring18/cos226/assignments/8puzzle/index.html

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning An Introduction second edition*. The MIT Press, 2018.

[4] H. M. Bastian Bischoff, Duy Nguyen-Tuong and A. Knoll, “Solving the 15-puzzle game using local value-iteration,” 2013.

[5] W. E. Johnson, Wm. Woolsey; Story, “Notes on the ”15” puzzle,” *American Journal of Mathematics*, vol. 29, no. 2(4), p. 397–404, 1879.

[6] D. of the Cube Forum, “5x5 sliding puzzle can be solved in 205 moves,” 2016. [Online]. Available: http://cubezzz.dyndns.org/drupal/?q=node/view/559

[7] Minsky, *Computation: finite and infinite machines.* Prentice Hall, 1967.

# Appendix A

# Project Planning Schedule

This is an appendix.

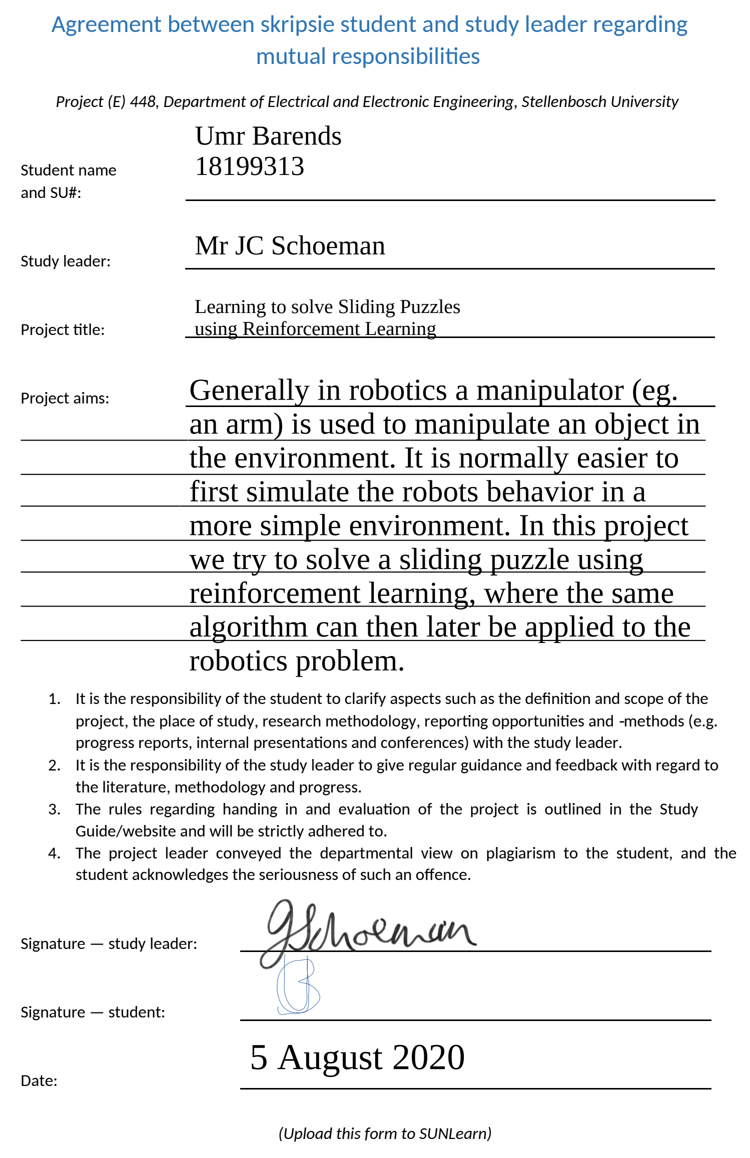
# Appendix B

# Outcomes Compliance

This is another appendix.

# Appendix C

# Student and Supervisor agreement



Agreement between skripsie student and study leader regarding mutual responsibilities

*Project (E) 448, Department of Electrical and Electronic Engineering, Stellenbosch University*

| | |
|---|---|
| Student name and SU#: | Umr Barends 18199313 |
| Study leader: | Mr JC Schoeman |
| Project title: | Learning to solve Sliding Puzzles using Reinforcement Learning |
| Project aims: | Generally in robotics a manipulator (eg. an arm) is used to manipulate an object in the environment. It is normally easier to first simulate the robots behavior in a more simple environment. In this project we try to solve a sliding puzzle using reinforcement learning, where the same algorithm can then later be applied to the robotics problem. |

1. It is the responsibility of the student to clarify aspects such as the definition and scope of the project, the place of study, research methodology, reporting opportunities and -methods (e.g. progress reports, internal presentations and conferences) with the study leader.
2. It is the responsibility of the study leader to give regular guidance and feedback with regard to the literature, methodology and progress.
3. The rules regarding handing in and evaluation of the project is outlined in the Study Guide/website and will be strictly adhered to.
4. The project leader conveyed the departmental view on plagiarism to the student, and the student acknowledges the seriousness of such an offence.

| | |
|---|---|
| Signature — study leader: | JSchoeman |
| Signature — student: | [illegible] |
| Date: | 5 August 2020 |

*(Upload this form to SUNLearn)*

**Figure C.1:** Student and Supervisor agreement