Supplementary Materials

Interactive Reinforcement Learning; Two successful solutions for handling an abundance of positive feedback

Jasper van der Waa, Maurits Kaptein, Khiet Truong

June 30, 2015

This document contains the appendices of the research “Interactive Reinforcement Learning; Two successful solutions for handling an abundance of positive feedback". Appendix A provides the reader with additional background knowledge. This appendix contains a more elaborate explanation of the term “return" in reinforcement learning, how agents can construct a policy from state-action values, the effects of the discount parameter on return and an explanation of the function approximation technique. Appendix B provides detailed explanations of the learning algorithms used in the study. Appendix C explains the TAMER framework in more detail, with a focus on the feedback distribution and how it was implemented in the agents. Appendix D provides screenshots of the entire experimental procedure and Appendix E contains the raw data of the measurements reported in the article. Finally, Appendix F contains some additional results.

Contents

Appendix A

Reinforcement Learning

A.1 Return; Discounted sequence of rewards

The entire reward of an arbitrary experienced state-action pair sequence from $t = 0, 1, \ldots n$, and starts in s_0, is defined as;

$$ValueSequence = \sum_{t=0}^{n} R(s_t, a_t, s_{t+1}) \tag{A.1}$$

This is the sum of all received rewards from R starting at $t = 0$ to $t = n$. The goal of RL agents is to maximize this return over an infinite amount of future steps; thus $n = \infty$ in Equation A.1. We do not want the agent to first experience an infinite amount of rewards before it knows how much reward a specific state-action pair sequence returns. Therefore these rewards are discounted by γ such that only a limited number of rewards are summed. The result is the cumulative discounted reward signals starting in s_0 and following some arbitrary known state-action pair sequence of infinite length. From here on this sum is referred to as the return of that arbitrary sequence;

$$Return(s_0) = \sum_{t=0}^{\infty} \left[\gamma^t R(s_t, a_t, s_{t+1})\right] \tag{A.2}$$

The agent's learning task is to learn this return based on a limited number of received reward signals regulated by the infinite state-action pair sequence. If the agent needs to actually experience all possible reward signals in this infinite chain, it will, obviously, go on for ever. Therefore the agent must learn to estimate this return. Whether this is actually possible depends on several aspects which is explained in Section A.2.

Up to know the return defined by Equation A.2 was based on an arbitrary known state-action pair sequence. However, we want the agent no to learn an estimated return for such an arbitrary sequence, but for sequences defined by some policy π. This policy determines the sequence of state-action pairs an agent experiences from any given starting state. However, as stated before a policy π is probabilistic in nature; from the same starting state, the agent can experience different state-action pair sequences. This has to be taken into account and from there we can define return for some specific policy π as follows;

$$Return(s_0, \pi) = \sum_{t=0}^{\infty} \left[\gamma^t \sum_{s_{t+1} \in S} (T(s_t, \pi(s), s_{t+1}) \cdot R(s_t, \pi(s_t), s_{t+1})) \right] \tag{A.3}$$

The return of some policy π starting in s_0, is equal to the sum of discounted reward of each experienced state-action pair, just like before. However, now these rewards depend onπ. This reward is now the sum of all possible next states s_{t+1} as determined by the transition probabilities given by T. Since this return is now more of an average of what cumulative rewards can be expected, we speak of expected return under some policy π. Of course, if π is a deterministic policy the Equation A.3 denoting expected return can be simplified to;

$$Return(s_0, \pi) = \sum_{t=0}^{\infty} \left[\gamma^t R(s_t, \pi(s_t), s_{t+1}) \right] \tag{A.4}$$

Whether π is deterministic or not, we do not differentiate between the expected return under a regular policy or simply return under a deterministic policy. To do this we write the reward the agent can expect from performing $\pi(s_t)$ in some state s_t, as a function E;

$$E_\pi \left[\gamma^t R\left(s_t, \pi(s_t), s_{t+1}\right) \right] = \gamma^t \sum_{s_{t+1} \in S} (T(s_t, \pi(s), s_{t+1}) \cdot R(s_t, \pi(s_t), s_{t+1}))$$

Often, the expected return is stated as the value of the starting state s_0. The expected return describes how valuable it is for the agent to be in s_0 with respect to the future rewards that can be collected under the policy π. In the original study however, we separate a state's value into the value of doing a specific action in that state. This gives the agent the option of selecting actions in a state based on the value of doing that specific action in that state. Otherwise the agent would have to choose its actions based on how likely it is that the agent will transition from its current state to a new state with a high value.

The value, in terms of expected return, of performing some action in a state is denoted by a function Q^π for a specific policy π; $Q^\pi \ : \ S \times A \to \mathbb{R}$. Here $Q^\pi(s, a)$ is the expected return when the agent choose action a in state s and follows policy π. More specifically;

$$\begin{aligned} Q^\pi(s, a) &= R(s_0, a, s_{t+1}) + Return(s_{t+1}, \pi) \\ &= R(s_0, a, s_{t+1}) + \sum_{t=1}^{\infty} E_\pi \left[\gamma^t R\left(s_t, \pi(s_t)\right) \right] \end{aligned} \tag{A.5}$$

The function Q is called the action-value function, as it provides the value in terms of expected return for taking an action in a specific state. The outcomes of the function Q are referred to action-values.

A.2 Learning Q and deriving policies

We desire of a RL agent that it finds a policy that actually maximizes the reward signals. Such a policy would give the action in every state that result in the highest rewards over time. Such an optimal policy is referred to as π^* for which $Q^{\pi^*}(s, a) \geq Q^\pi(s, a)$ for any other policy π and $\forall a \in A$ and $\forall s \in S$.

The mechanism an agent uses to derive such a policy from action values is by greedily choosing the action that has the highest action value; $\pi(s) = maxarg_a\left(Q^\pi(s,a)\right)$.

For an agent to find π^*, its action-value function needs to output the true action value stated by Equation A.5. However, initially, the agent does not know what kind of reward signals the reward function R outputs. The agent needs to experience these and adjust its initial Q^π to a new Q^π' that reflects these experienced reward signals better.

An agent can update its Q^π based on knowledge about π or not. If the updates do depend on π, the agent learns using a so called on-policy method. This means that the agent uses the assumption that π is used to select actions and determines what the agent will experience in the future. If an agent updates Q^π independent of any knowledge of π, the policy currently being followed, the agent uses a so called off-policy method for learning. This means that the learning algorithm of the agent does not use π to define the expected return as given by Equation A.5. The algorithms used in the study are all off-policy methods. For this reason we often omit mentioning π in Q^π, thus Q^π can be referred to as simply Q (and $Q^\pi(s,a)$ as $Q(s,a)$).

We note here that off- and on-policy methods only refer to whether the policy the agent currently follows, the behavior policy, is used in updating Q or not. Off-policy methods can still derive some policy from Q (to define a next action for example) and use it in its update mechanism. As long as the action values are updated independently off the actual behavior policy, the learning method remains an off-policy method. For example, a policy can be derived from Q using the greedy action selection mechanism; $\pi(s) = maxarg_a\left(Q(s,a)\right)$. This allows the update mechanism of an off-policy method to use the policy that is greedily derived from its current Q independent of the actual behavior policy (which makes it off-policy).

If the action-value function reflects the true discounted sum of returns from Equation $A.5$, we refer to Q as the optimal value-function Q^*. The policy derived from Q^* using the greedy action selection mechanism is the optimal policy π^*. Whether the agent is ever capable of deriving π^*, depends on the learning algorithm used that updates Q to Q'.

A.3 The values and effects of the discount factor

The discount factor, γ, affects the effective size of the infinite horizon stated in Equation A.5. It does this with a reason, as an infinite sum of values is either positive infinite, negative infinite or zero. The discount parameter makes sure that as the sum steps further into the future, the rewards are weighted less and less. At some point any additional rewards will contribute very little to the entire sum. However this is only the case when $0 \leq \gamma < 1$, which we explain below.

If $\gamma = 1$, then none of the future rewards are discounted as $\gamma^x = 1^x = 1$ for all $x \in \mathbb{N}$. At this point the Equation A.5 will diverge, which makes it impossible for the agent to estimate as the function would have no limit. To show this we will treat the infinite sum of expected returns stated in Equation A.5 as a mathematical series (known as a geometric series) by substituting $R\left(s_t, \pi(s_t)\right)$ with the temporary variable x and abstract from a specific policy to prevent clutter. This results in the equation;

$$Q = \sum_{t=0}^{\infty} E\left[\gamma^t x\right] \tag{A.6}$$

A geometric series like this, is said to converge when it approximates a specific value even though it is an infinite sum of values. To show this we first define a nth-partial sum of this infinite sum as; $S_n = \sum_{t=0}^{n} E\left[\gamma^t x\right]$. Now, imagine the nth-partial sum with a reasonable small n, for example $n = 10$,

so we get $S_{10} = \sum_{t=0}^{10} E\left[\gamma^t x\right]$, which gives some value for every γ. Now take a second nth-partial sum with a very large n, for example $n = 10000$, so we get $S_{10000} = \sum_{t=0}^{10000} E\left[\gamma^t x\right]$ which also has some value for every γ.

Now Q is only stated to converge, in this example, if and only if the difference between S_{10} and S_{10000} is arbitrary small. Lets call this difference ε, then we get;

$$\varepsilon = \mid S_{10} - S_{10000} \mid = \mid \sum_{t=0}^{10} E\left[\gamma^t x\right] - \sum_{t=0}^{10000} E\left[\gamma^t x\right] \mid = \sum_{t=10}^{10000} E\left[\gamma^t x\right]$$

If ε is indeed arbitrary small we say that our action-value function Q converges. Since increasing t from 10 to 10000 will only add the arbitrary small value ε to the value given by S_{10}. In other words, Q approximates the value S_{10} in this example. However this is only the case when $\gamma < 1$. As soon as $\gamma \geq 1$, the difference ε keeps increasing as t becomes larger since $\gamma^t \geq 1$ for all $t \in \mathbb{N}$. This does not assure us with convergence as the difference is not guaranteed to be small for high values of t. Thus in the above example, ε is not arbitrary small in as $S_{10000} \gg S_{10}$ if x does not decrease sufficiently over time (which is very unlikely as x is independent of t).

If the optimal action-value function Q^* does diverge because $\gamma \geq 1$, it means that any Q from the agent will only output infinity, negative infinity or zero for all state-action pairs. Thus there is no guarantee that any action based on the learned values are indeed the actions that result in the highest return. Therefore no learning algorithm can guarantee that the agent finds the optimal policy, not even after an infinite amount of learning. For this reason the discount factor γ will never be equal or greater then 1.

Finally, γ must always be positive otherwise the expected return $E_\pi\left[\gamma^t R\left(s_t, \pi(s_t)\right)\right]$ at t becomes negative when R gives a positive reward and vice-verso. If this would be the case, then every rewarding action would become a penalizing action. This, and the statements above, effectively sets the domain of γ to $[0, 1)$.

The use of the discount factor γ extends beyond making reinforcement learning possible. It is a parameter that defines the number of steps the agent "looks" into the future when deciding which actions are the most rewarding. More formal; γ is used to define Q such that it approximates the sum of rewards of a specific amount of future steps.

Take the previous example where Q approximated $S_{10}$and any larger partial sums would only differ very little with S_{10}. In this example the agent would take only the time steps from $t = 0$ to $t = 10$ into account when estimating the future consequences of some action in terms of reward. Any more steps would simply not change the expected return with a noteworthy amount.

If we know that Q converges to a specific value due to $\gamma \in [0, 1)$, we can calculate the exact number of k steps the agent "looks" into the future for any specific γ;

$$k = \frac{1}{1 - \gamma} \tag{A.7}$$

Or we can define γ, for a specific amount of k steps the agent should take into account;

$$\gamma = \frac{k - 1}{k} \tag{A.8}$$

In our example where Q approximate the expected return of the agent following some arbitrary policy for 10 steps ($k = 10$), the discount parameter was $\gamma = \frac{10-1}{10} = 0.9$.

From Equations A.7 and A.8 we can calculate how many steps the agent will look into the future when learning an optimal policy. Since an optimal policy is only optimal when it results in the max

possible amount of expected return, the γ value is usually set to values close to 1. Such values allow the agent to select actions that are indeed the most optimal as they result in high amounts of expected returns over time. Common values of γ are 0.8,0.9 and 0.999 that respectively take 5, 10 and $10\,000$ future time steps in to account.

Although a high discount factor makes it easier for the agent to find the actual optimal policy, it will also take more time. For example with $\gamma = 0.999$ the agent will take the future $10\,000$ state-action pairs into account. The agent still has to experience the reward signals from these state-action pairs at least once. Otherwise the agent may not learn a decent estimation of the expected return over that many time steps.

A special case in Reinforcement Learning is when $\gamma = 0$, meaning that $k = 1$ from Equation A.7. Any agent with this discount factor will only take the immediate expected return into account when taking an action. Such agents will choose actions that result in the immediate highest reward signal from the reward function R. These agents will be relatively immune to changes in R. The learned Q of these agents is easily adjusted. This compared to an agent with a high γ. Such an agent has to re-experience a large set of state-action pairs again to account for any changes in R. The downside of agents with $\gamma = 0$, is that their is almost no chance of them finding the optimal policy. These agents greedily select the current best actions without any regard of what the future may as a result of those actions. At best these agents settle in a sub-optimal policy. There is only the slightest chance that agents with $\gamma = 0$ ever happen to stumble over the optimal policy. Therefore, reinforcement learning agents have almost never their discount parameter set to zero.

A.4 Function Approximation

It was repeatedly stated that a RL agent tries to learn estimations of the expected return for taking an action a in state s; the $Q(s, a)$ values. The agent defines some policy π such that the mapped actions, maximize these estimations.

Up to now it was not discussed how these $Q(s, a)$ are stored in the agent. The most simple method would be storing them in a large table of size $\mid S \mid \times \mid A \mid$. The problem with this method is that if there are a lot of possible states, so $\mid S \mid$ is large, memory becomes a problem. This method also separates every possible state-action pair and treats them independent from each other. The result is that the agent is not capable of generalizing any of its learned $Q(s, a)$ over similar state-action pairs.

A solution to these two issues, is to approximate the $Q(s, a)$ values with a linear function f, defined as; $f : S \times A \rightarrow \mathbb{R}$. This linear function f uses two vectors to approximate Q; $\vec{\phi}$, the feature vector and $\vec{\theta}$, a weight vector.

The feature vector $\vec{\phi}$, of length n, is constructed from the function $\phi : S \times A \rightarrow \mathbb{R}^n$. This function receives a state s and action a as input, and constructs a specific set of features that is unique to that state-action pair. A specific feature vector of state-action pair (s, a) is referred to as $\phi(s, a)$ in its functional form or $\vec{\phi}_{s,a}$ as the function's outcome. We note here that the features themselves do not solely have to depend on the received state-action pairs. For example, features can be constructed from the distance of a state s to all other states $s' \in S$.

The weight vector $\vec{\theta}$ is a vector of values that the agent can adjust such that $f(s, a) = \vec{\phi}_{s,a}\vec{\theta} \approx Q(s, a)$. These weights are usually updated using regression methods, such as incremental gradient descent, that try to minimize an error function. Such regression methods are usually integrated in the update function(s) of the RL algorithm. These regression methods are not discussed in detail here. For a detailed description of regression methods for function approximation in RL, we refer to "Reinforcement Learning; An Introduction" by Barto (1998).

The approximation function f expresses the action-value function Q, as the linear function $Q(s,a) \approx \vec{\phi}_{s,a}\vec{\theta} = f(s,a)$. From here on, we assume that f substitutes Q in RL algorithms that use (linear) function approximation. Under this assumption the agent will update $\vec{\theta}$ such that the $f(s,a)$, thus also $Q(s,a)$, approximate $Q^*(s,a)$ as more state-action pairs are experienced.

This function approximation method allows the agent to generalize over state-action pairs. In the tabular method, if the agent experiences a new state-action pair the table would only provide some initial value. However, with function approximation this new state-action pair is transformed into a feature vector and multiplied by the learned weight vector. This allows the agent to use its learned estimations expressed by the weight vector, to already formulate a more sensible action-value for that new state-action pair. Also storing a single vector is much memory efficient then storing all possible state-action pairs.

Appendix B

TAMER Framework

The TAMER framework is a framework that defines several important design issues for interactive reinforcement learning (IRL) agents (Knox, 2012). TAMER stands for "Training an Agent Manually via Evaluative Reinforcement". This framework consists of three modules; the feedback distribution module, the reward model and the action selector. These three modules combined describe an interactive reinforcement learning agent that addresses several important issues with such agents. See Figure B.1 for an overview of how these three modules form an agent and the interaction between them, the environment, and the user.

The task of the feedback distribution module is to credit the appropriate (s, a) with a provided feedback value F. This decision of which transition tuple deserves how much credit from F is done with the usage of the user's modeled reaction time. The credit of state-action pairs weighs F to form $H(s, a)$, the actual user defined reward signal. Thus the feedback signals are not the actual reward signals. This module credits certain (s, a) for causing F based on the probability a user could respond to to (s, a) on time with F.

The reward model receives the reward signals from H. The model learns to label state-action pairs with a robust average of the received reward signals paired with those state-action pairs. These modeled signals and are referred to as $\hat{H}$, the modeled user reward function. This allows the agent to define a reward signal for a state-action pair even if one is not provided by the user. The model also provides a method in creating a more constant reward function based on the inconsistent feedback values from user.

Finally, the action selector receives the reward signals from $\hat{H}$ and uses them in a reinforcement learning algorithm as a substitute of R. In the sections below we will explain each module in more detail.

B.0.1 The Feedback Distribution Module

The purpose of the feedback distribution module is to determine which past state-action pairs should be credited with how much of F_t. The module does this by first calculating this amount of credit. This credit is the probability that (s, a) may have been targeted by the user when he gave F_t. The module determines this probability or credit based on a model of the user's reaction time. This means that the probability some state-action pair is targeted with some F_t, is based on how fast the user responds to a perceived state-action pair with feedback. Therefore this module assumes that the user will provide the agent with F_t as soon as the targeted state-action pair is perceived and assessed.

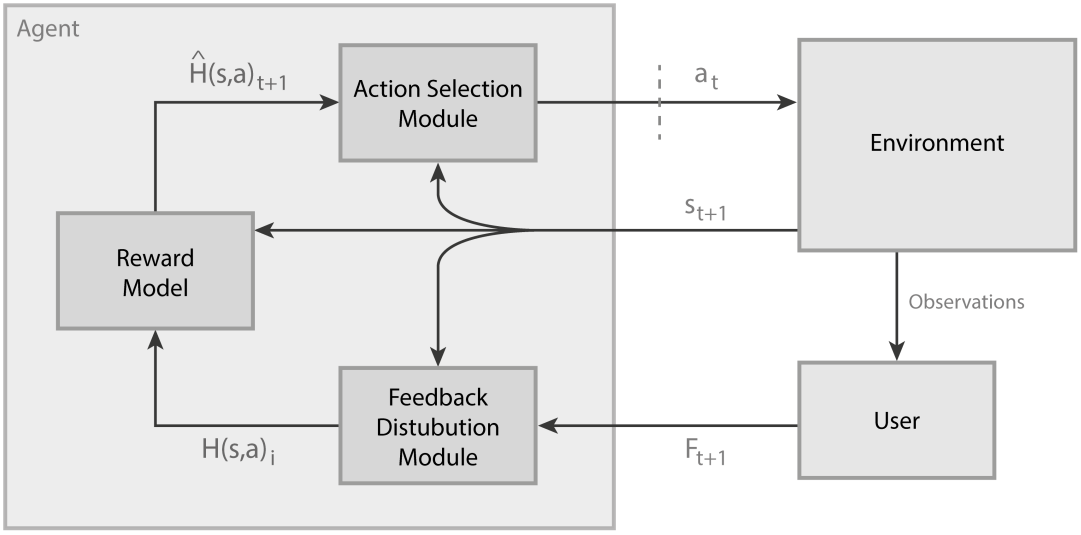


Figure B.1 – An overview of the inner workings of a TAMER agent and its interaction with the environment and its user. The agent performs an action which changes the environment. This new environmental state is perceived by the agent and the user observes the made changes. Based on these observations, the user assessed the agent's action or overall behavior and provides the agent with a feedback signal. This feedback signal arrives at the Feedback Distribution Module who spreads the value of the feedback signal over past states according to a model of the user's reaction time. This module outputs a reward signal for an arbitrary state-action pair that cannot receive any more feedback in the present and future. This reward signal is a summary of all the feedback values that were distributed to it according to the given feedback signals and reaction time model. The reward signal is given to the Reward Model, whose task it is to model what reward signals belong to which state-action pairs. This model outputs a modeled reward signal for every experienced state-action pair. This modeled reward signal is used by the Action Selection Module as a regular reinforcement learning reward signal in a learning algorithm. This algorithm learns to optimize its action choices with respect to these modeled reward signals.

The model of the user's reaction time takes the form of a probability density function referred to as f. This function f defines the probability that a past state-action pair could be targeted by the user based on his reaction time. For example, the higher the probability that very recent state-action pairs are targeted, the faster the user is expected to respond with feedback values.

This module allows the agent to deal with the delay between the experienced state-action pair $(s,a)_i$ and the given feedback F_t. Also, this module allows some freedom in how this delay should be treated though the definition of f. For example, f can be a gamma function or a simple uniform distribution. The first distribution will give more credit to the most recently experience state-action pairs due to its shape. While the uniform distribution will give equal credit to all state-action pairs that fall within the non-zero area of f.

Although some experienced state-action pair $(s,a)_i$ was first experienced at time i, the pair still took a small time interval of size n. Where n is defined by the time it takes the agent to process the perceived state, to learn, select the next action, and perform the action. This means that $(s,a)_i$ may

have started at time i but continued to last till $i + n$ where a new state-action pair $(s', a')_{i+n}$ was experienced.

Below a formal definition of the credit c_i^t is given. Where c_i^t is the credit for some $(s, a)_i$ experienced first at time i, lasting till $i + n$, when the user provided F_t at time t;

$$c_i^t = P\left(targets(F_t,\ (s,a)_i)\right) = \int_{i-t}^{i+n-t} f(x)dx \tag{B.1}$$

Here the integral of f, is used to calculate the probability that $(s, a)_i$ receives any credit for causing F_t. This integral is over the interval $[i - t;\ i + n - t]$ which is the time interval of (s, a) relative to the time t at which F_t was given by the user. See Figure B.2 for an example. This example shows how $(s, a)_i$ is credited when the user provides the agent with F_t, according to a uniform probability distribution f.

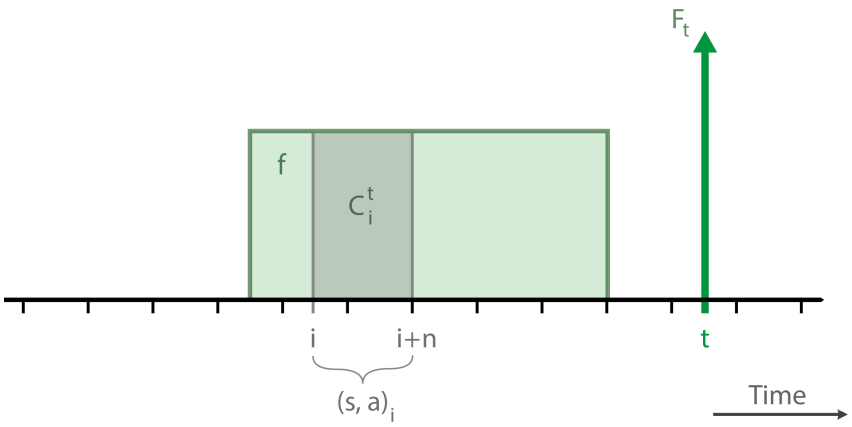


Figure B.2 – An example of where a reaction time model f, here a uniform distribution, targets a specific state-action pair $(s, a)_i$. The surface of f from i to $i + n$ is the amount of credit assigned to $(s, a)_i$, referred to as c_i^t.

With a known credit c_i^t the amount of credited feedback, $H(s, a)_i^t$, can be calculated for $(s, a)_i$ for the given F_t. This is done by weighing F_t with the credit c_i^t to get the amount of feedback (s, a) is held responsible for, according to f;

$$H(s,a)_i^t = P\left(targets(F_t,\ (s,a)_i)\right) \cdot F_t = c_i^t \cdot F_t$$

However, the user can give multiple feedback signals to the agent in a short period of time. If this is the case, it can very well occur that, according to f, a state action pair $(s, a)_i$ is credited for more than one feedback signal. See Figure B.3 for an example of such a situation. In this example the user gave feedback at times $t1$ and $t2$ and the state action pair $(s, a)_i$ should receive credit for both according to f.

In such a situation we simply sum all $H(s, a)_i^t$ for every time t at which a feedback signal was received. Although this is a simple solution it requires the module to wait until $(s, a)_i$ can never receive any more credit from a feedback signal. This is visualized in Figure B.4. Here, the state-action

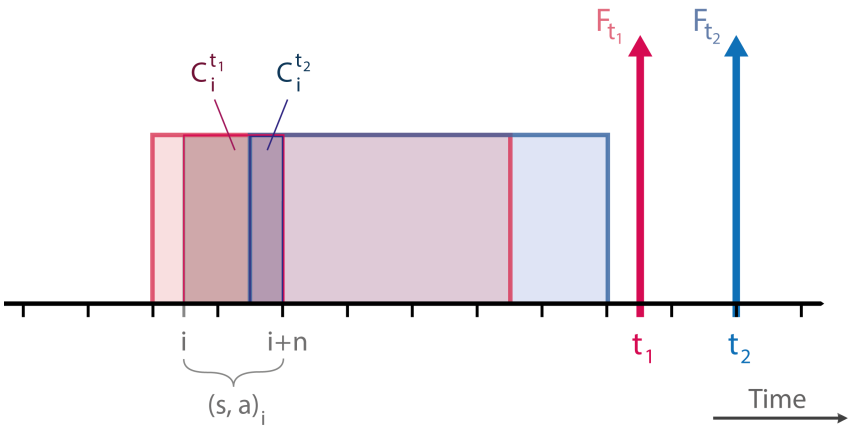


Figure B.3 – An example where the user provided two feedback signals, F_{t_1} and F_{t_2}, at times t_1 and t_2 respectively. The reaction time model f is used for each feedback signal to determine the amount of credit for state-action pairs. In this case the state-action pair $(s,a)_i$ overlaps with the f cast from F_{t_1} and overlaps partially with the f cast from F_{t_2}. Hence, $(s,a)_i$ is credited twice with $c_i^{t_1}$ and $c_i^{t_2}$ (with $c_i^{t_2} < c_i^{t_1}$). Therefore the eventual reward signal for state-action pair $(s,a)_i$ consists out of the sum of two feedback signals weighted by their respective credits $c_i^{t_1}$ and $c_i^{t_2}$.

pair $(s,a)_i$ lies outside the boundaries of the uniform distribution f that starts at the present time. Thus the integral of f over the interval $[i,\, i+n]$ will always be zero; $(s,a)_i$ cannot be credited anymore for any possible current or future feedback value.

To be able to sum all $H(s,a)_i^t$ for all given F_t after $(s,a)_i$ cannot receive any more credit, the agent needs to store a history of F_t. We refer to this history of all received feedback signals as F_{hist}. This F_{hist} contains all feedback values received by the agent up to the present. Now a formal definition can be given for $H(s,a)_i$, the reward signal for $(s,a)_i$ based on all past feedback signals;

$$H(s,a)_i = \sum^{F_t \in F_{hist}} H(s,a)_i^t = \sum^{F_t \in F_{hist}} F_t \cdot c_i^t = \sum^{F_t \in F_{hist}} \left[F_t \cdot \int_{i-t}^{i+n-t} f(x)dx \right] \tag{B.2}$$

For this module to function, the agent has to memorize both a history of feedback signals (F_{hist}) and state-action pairs (referred to as $Pair_{hist}$). It needs the history for state-action pairs to recall the action-pair that cannot be credited any more. However, this history can be limited to all state-action pairs that occurred after t_{ε_c}, as all older pairs already have their reward signals defined. This limited history of state-action pairs can then be used to limit the history of feedback signals. There is no point in storing a feedback signal that occurred at some point in time when all the previous state-action pairs already have a defined reward signal.

See Algorithm B.1 for how this module was implemented in the agents used in the study. This algorithm also provides an overview of the steps this module follows when defining H. Finally, it shows how the two history sets F_{hist} and Pa_{hist} are used and updated.

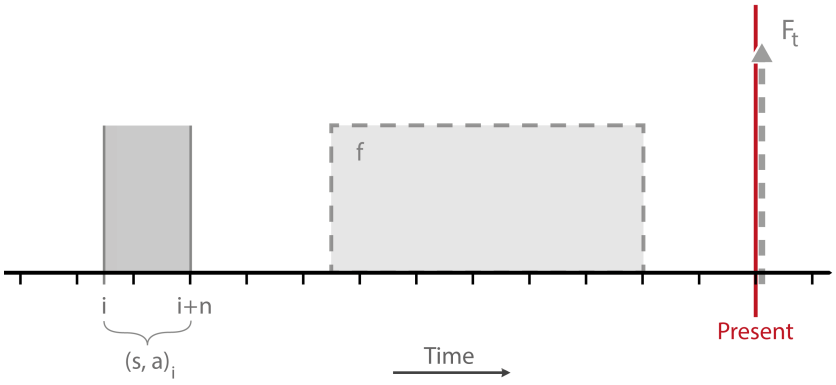


Figure B.4 – An example where the agent is at the present time and any feedback signal provided at this time (or the future) casts an f that may assign $(s,a)_i$ with any credit anymore. This is simply because the end time of $(s,a)_i$ happened before the start time of this possible f. In such a case the state-action pair $(s,a)_i$ can be forgotten from the agent's memory and its reward signal is also fully defined as it cannot change anymore (when this happens the $H(s,a)_i$ is send to the reward model).

Algorithm B.1 The algorithm behind the Feedback Distribution Module.

begin
var t_c, f
  for all $(s,a)_i$ **in** $Pair_{hist}$
    if $i < t_{\varepsilon_c}$
      $H(s,a)_i \leftarrow \sum^{F_t \in F_{hist}} \left[F_t \cdot \int_{i-t}^{i+n-t} f(x)dx \right]$
      $Pair_{hist} \leftarrow Pair_{hist} \backslash (s,a)_i$
    end if
  end for
  for all F_t **in** F_{hist}
    if $\int_{i-t}^{0} f(x)dx = 1$ for all $(s,a)_i \in Pair_{hist}$
      $F_{hist} \leftarrow F_{hist} \backslash F_t$
    end if
  end for
end

Appendix C

Algorithms

This section discusses the main learning algorithms used in the performed study. Two reinforcement learning algorithms are discussed and one regression algorithm. These three algorithms are explained with some level of abstraction to clearly state the underlying idea of each. This means that their learning mechanisms are explained but a theoretical proof that they indeed work is not provided. The explanation of the two reinforcement learning algorithms will use the notation and concepts introduced in the previous section. The regression algorithm uses the notation common in the field of machine learning.

C.1 Q-Learning

The Q-Learning algorithm is one the oldest reinforcement learning algorithms (Watkins, 1989). It is the first reinforcement learning algorithms that allowed the agent to learn action-values; the value of a performing a specific action in a state opposed to just the entire value of one state. It comes as no surprise that the Q-Learning algorithm is named after learning these action-values for state-action pairs for any given policy; $Q^{\pi}(s, a)$.

C.1.1 Bellman Optimality Equation

The Q-Learning algorithm is built from the Bellman optimality equation that defines Q^* from the expected return definition as stated in Equation A.3. The Bellman optimality equation is;

$$Q^*(s,a) = \sum_{s' \in S} T(s,a,s') \left[R(s,a,s') + \gamma \underset{a'}{max} Q^*(s',a') \right] \tag{C.1}$$

The value of doing some action a in s, is the return that the agent receives in every possible state s' the agent might find itself in after performing a, weighted with the (known) $T(s,a,s')$ that gives the probability of $P(s' \mid s,a)$. The return an agent receives when a brings the agent from s in some s' is equal to the actual reward received from that transition plus the potential of s'. Where this potential is described as the highest action value possible in s' (discounted by γ).

Equation C.1 describes the optimal Q^* because the equation incorporates the greedy action selection mechanism provided in Section A.2. This mechanism is a way to infer a policy from a Q such that the policy is optimum according to the values stated by that Q. Thus, Equation C.1 provides a

definition of Q^* based on the fact that the highest expected return of future state-action pairs is equal to the rewards received while performing the actions that maximize the action values.

Note from this equation that it does not depend on any knowledge of the behavior policy itself. Hence, all algorithms that use this equation to define Q^* use an off-policy method. This includes the Q-Learning algorithm.

C.1.2 Q-Learning; The Update Rule

The Q-Learning algorithm tries to learn Q^*; it updates its own action-value function Q based on experienced rewards, the definition of expected return (Eq. A.3) and the Bellman optimality equation (Eq. C.1). With each experience reward signal, Q is updated to give state-action values closer to the values that should be outputted by the defined Q^*. However, the challenge is that the agent does not know Q^*, otherwise it would have nothing to learn.

From $Q(s,a)$ we expect that it describes $R(s,a,s') + \gamma max_{a'} Q(s',a')$ as stated by what the optimal action-value function should described according to the Bellman optimality equation (Eq. C.1). The Q-Learning algorithm formulates its estimation error of $Q(s,a)$ not based on the difference between $Q(s,a)$ and $Q(s,a)^*$ since $Q^*$is not known. The algorithm solely bases its estimation error difference between its estimate and the experienced reward plus the estimated potential or value of the next state. More formal; the estimation error of the Q-Learning algorithm is the difference between $Q(s,a)$ and $R(s,a,s') + \gamma max_{a'} Q(s',a')$. Even if $Q(s',a')$ is an inaccurate description of the expected return of performing a' in s', we still expect $Q(s,a)$ to describe this (erroneous) estimation plus the experienced reward signal. Therefore, any estimation error of $Q(s,a)$ is because it was not able to describe the reward signal $R(s,a,s')$ appropriately. Since $R(s,a,s')$ is experienced immediately after performing a, the algorithm can thus formulate an estimation error of $Q(s,a)$ after performing a. Any possible error in $Q(s',a')$ is a problem for possible the next step where the agent might perform a' in the new state s'. As the agent continuous to perform actions in states, it will experience reward signals, formulate estimation errors and updates its action-value function accordingly. This means that from updating one action value, the entire action-value function is improved. For example, if $Q(s',a')$ is updated to be more accurate, then $Q(s,a)$ will also become more accurate since it depends on $Q(s',a')$.

A formal description of the estimation error is called the temporal-difference error, referred to as δ;

$$\delta = \Big[R(s,a,s') + \gamma \max_{a'} Q(s',a')\Big] - Q(s,a) \tag{C.2}$$

It is called a temporal-difference error because it describes the difference between an estimation and the actual outcome over a change from state s to s' in time. As explained above, any non-zero δ reflects an error in the prediction $Q(s,a)$ regarding $R(s,a,s')$, since $Q(s',a')$ is known and treated to be fully accurate for this increment.

Since δ describes the error the agent made with $Q(s,a)$, this value can be updated to a more accurate $Q'(s,a)$. This is done by simply weighting the error δ with a parameter α, the learning rate. The learning rate is a small positive value defined by the agent's designer. This results in the update rule for Q-Learning;

$$\begin{aligned} Q'(s,a) =& Q(s,a) + \alpha\delta \\ =& Q(s,a) + \alpha\Big[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big] \end{aligned} \tag{C.3}$$

If the agent repeatedly experiences (s,a), this update rule assures that $Q'(s,a)$ will resemble its actual value better. This actual value is;

$$Q(s,a) = \sum_{s' \in S} T(s,a,s') \left[R(s,a,s') + \gamma \underset{a'}{max} Q(s',a') \right]$$

Since, repeatedly experiencing (s,a) is sampling different s' and $R(s,a,s')$ from $T(s,a,s')$. If $Q(s',a')$ is equal to the optimal $Q^*(s,a)$, this equation is equal to the Bellman optimality equation shown at Eq. C.1. Again, if $Q(s',a')$ also gets increasingly through sampling the $Q(s,a)$ also becomes more accurate without needing any extra updates. As this continuous for all state-action pairs, the Q can converge to the optimal state-action value function Q^* (Barto, 1998).

C.2 TDC Learning

One of the downsides of Q-Learning is its lack of generalization of learned estimations over different state-action pairs. We could use function approximation to represent the look-up table of Q-Learning and enforce generalization this way, as discussed in Section A.4. However, if we apply this method directly to Q-Learning, it will cause Q to diverge when applied to learning algorithms such as Q-Learning. This is because these algorithms update their action values without any regard to the changes the update makes to the other action values.

This effect is best explained with an exemplar MDP called the "Star Problem" (Baird, 1995). This MDP is shown in Figure C.1. The Star Problem contains seven states and the value of each state is given by the linear combination of two weights. A value of some state s is denoted here with $V(s)$, with $V(s) = max_a Q(s,a)$. Every transition from one state to another results in a reward signal of zero. Each possible transition is observed by the agent equally often. This function approximation method is a viable on for this problem; values are described linearly and it can generalize (all values can be described and all values use at least one shared weight). Also, with the update of one value, other are affected as well since all states are connected with at least one other state. Therefore you could say that this system has to converge. As all states are experienced equally often and weights are appropriately updated, the values have to converge to some value (zero in this case, as all rewards are zero). Any new update would only make the weights to cause a greater error for some state. However, this is not the case.

In the Star Problem example, the weights and value would converge to zero if w_0 was not used. In this case the weights would simply act as a look-up table. However, if all weights are initially set to positive values and $V(7)$ is much larger than all other values, all values grow to infinity. This is due to the fact that the first six states all have a value smaller then their successor, $\gamma V(7)$. Also the value of state seven has itself as its successive state therefore its value will incorporate its own discounted value $\gamma V(7)$. Since $\gamma < 1$ we know that $\gamma V(7) < V(7)$. Thus we know that $V(1 \text{ to } 6) < \gamma V(7) < V(7)$, and $\gamma V(7)$ needs to be a part of very state value. While updating, this means that the shared weight w_0 will increase six times (to account for $V(1 \text{ to } 6) < \gamma V(7)$) for every time it is decreased (to account for $\gamma V(7) < V(7)$). Of course w_7 will decrease as well but too slow, since state 7 is visited equally often as all other states. This results in all values and weights to increase indefinitely.

Although the Star Problem is a very specific problem, it does show how function approximation can prevent convergence when the learning algorithm does not account for the errors a shared weight update causes in other values. Also, this issue is not confined to the Star Problem alone. It can occur in every MDP where the balance between values is so that a shared weight is increased (or decreased) multiple times for every time the weight is decreased (or increased).

Therefore, to allow generalization of state-action value estimations, we need an algorithm that learns in a way that includes the possible errors a shared weight update causes in the states that

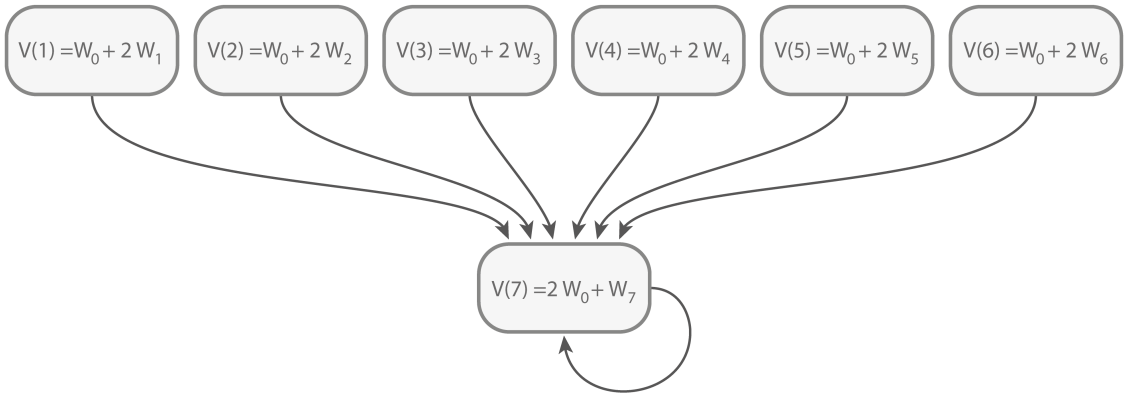


Figure C.1 – The Star Problem. An example of an MDP in which a reinforcement learning such as Q-Learning is most likely to converge if the technique of function approximation simply substitutes the state-action value look-up table. The figure is adapted from Baird (1995). In this problem the shared weight w_0 and the fact that $V(7)$ is reachable from any state, including itself, causes Q-Learning to diverge as w_0 is increased (or decreased) six times more often then it is decreased (or increased). This occurs when all states are visited just often, all reinforcement is zero and $V(7)$ is initialized with a slightly higher value then the others.

share this weight. One of these algorithms is the TDC algorithm; "Temporal Difference with gradient Correction". This algorithm is more complex then the Q-Learning algorithm and a formal description is beyond the scope of this chapter. A formal overview of this algorithm is given by Sutton et al. (2009a) and the necessary background knowledge by Baird (1995) and Sutton et al. (2009b).

The difference between the TDC Learning algorithm and the Q-Learning algorithm is the different loss function each tries to minimize. The Q-Learning algorithm tries to minimize the Mean Squared Bellman Error (MSBE). The TDC Learning algorithm minimizes the Mean Squared Bellman Projected Error (MSBPE). The MSBE is the (mean squared) difference between a prediction $Q(s, a)$ and the experienced reward $R(s, a, s')$ plus the value of the next state $\gamma \underset{a'}{max} Q(s', a')$. This loss function is the basis of the definition of the temporal difference error shown in Equation C.2. Note that it does not account for any weight sharing; a function approximation method would simply replace $Q(s, a)$ with the approximation $f(s, a)$. This is not desired, as we want a learning algorithm that accounts for the fact that function approximation is used. We want the learning algorithm to know and use the fact that some or all weights are shared. This allows the learning algorithm to adjust the weights such that the perceived approximation error is minimized while not creating additional errors in different state-action values. The MSBPE loss function describes a method of doing this. It takes the weight vector and the estimation error calculated based on the current experienced reward signal. Then this loss function does not simply state that the weights needs to be adjusted such that this error is reduced. This loss function states that the estimation error needs to be reduced while minimizing the change in all other values that the weights describe. This ensures that any learning algorithm based on MSBPE is not capable of adjusting weights to minimize the estimation error at the cost of a large change in the values described by these (shared) weights.

TDC Learning, which is based on MSBPE, can use a weight vector to approximate its predictions as a linear function and still be able to converge into an action-value function (Baird, 1995; Sutton et al., 2009a). Although other algorithms exist that can do the same, the TDC Learning algorithm has $O(n)$

complexity opposed to other algorithms with $O(n^2)$ or higher, with n the size of the feature vector $\vec{\phi}$. To conclude, the TDC Learning algorithm is a state of the art algorithm that converges, learns fast and is efficient, while using function approximation (Sutton et al., 2009a). This allows the algorithm to generalize its predictions over multiple state-action values. The algorithm allows an agent to use its learned knowledge from one experience to affect the knowledge learned in the past and future.

C.3 Incremental Gradient Descent

At the basis of numerous machine learning techniques lies the search for a some linear function g. This g takes a feature vector $\vec{x}$ and a weight vector $\vec{w}$ as input. It outputs some y that is close or equal to some known value that belongs to $\vec{x}$. The challenge is to adjust $\vec{w}$ such that the labels outputted by g become more accurate. The accuracy of the weight vector is measured by a loss function; $L(\vec{w})$. The Incremental Gradient Descent (IGD) algorithm is one of the machine learning techniques that uses this approach.

The IGD algorithm minimizes $L(\vec{w})$ in a relative simple way. This algorithm processes one training sample at a time. Each sample consists of a feature vector and known label. With each sample the algorithm updates its weights $\vec{w}$ into a new set of weights $\vec{w}'$ such that the error from the objective function decreases. The IGD algorithm takes the derivative of L to calculate the gradient of the made error. This gradient is used $L(\vec{w}') < L(\vec{w})$. This results in the following update rule;

$$\vec{w}' = \vec{w} - \alpha \nabla L_i(\vec{w}) \tag{C.4}$$

In this equation, $L_i(\vec{w})$ is the error caused by $\vec{w}$ on the ith training sample and $\nabla L_i(\vec{w})$ is the gradient (or direction) of this error. The learning rate or step size α prevents to affect $\vec{w}$ to much with one update. Since if $\vec{w}$ would change too much at each sample, the algorithm will not be able to converge into a set of weights that minimizes L for all samples.

The performed study uses IGD with a the least squares loss function and g a linear function of $\vec{x}$ and $\vec{w}$. The error for some $\vec{w}$ on the ith training sample is then defined as;

$$L(\vec{w}) = \frac{1}{2}\left(y_i - g(\vec{x}_i, \vec{w})\right)^2 = \frac{1}{2}\left(y_i - \vec{x}_i^\top \vec{w}\right)^2$$

This objective function combined with the Equation C.4 results in this specific update rule;

$$\vec{w}' = \vec{w} - \alpha \nabla \frac{1}{2}\left(y_i - \vec{x}_i^\top \vec{w}\right)^2 = \vec{w} - \alpha\left(y_i - \vec{x}_i^\top \vec{w}\right)\vec{w}$$

If the learning rate α decreases over time and that the samples are randomly drawn from a distribution, the IGD algorithm is guaranteed to converge to a local minimum (Knox, 2012). For the linear functions used in the study, it is even guaranteed that IGD converges to the global minimum.

Appendix D

The experiment procedure

This appendix provides a series of screenshots from the experiment, including a translation of the Dutch instructions in each screenshot.

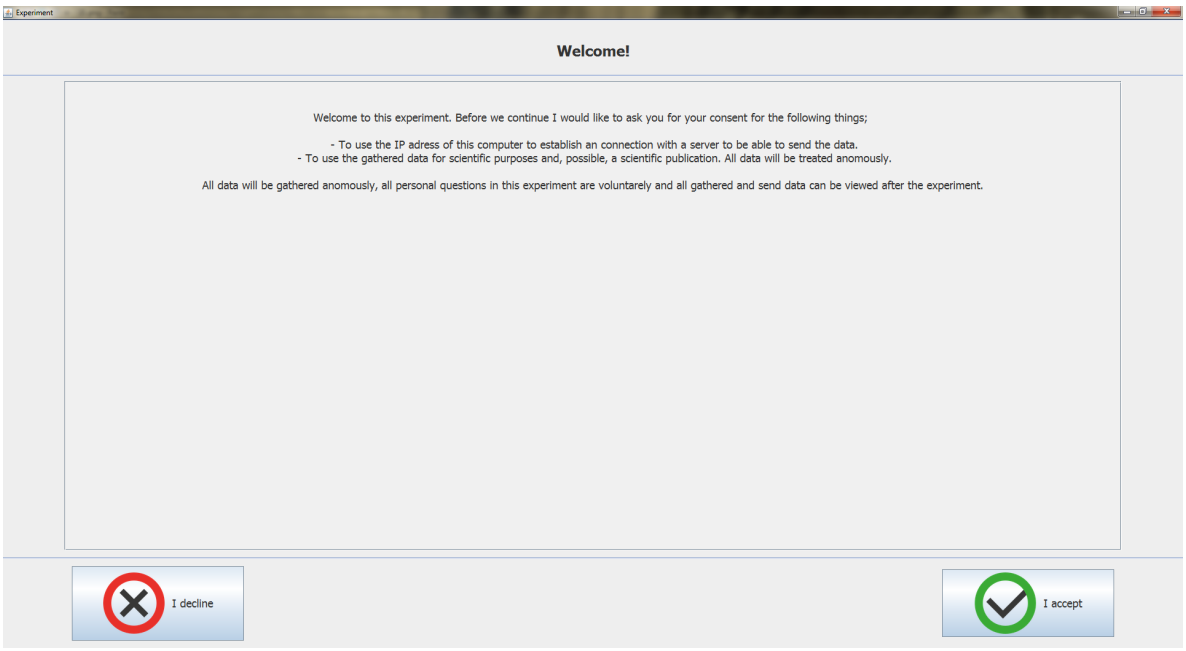


Figure D.1 – First screen; Consent form.

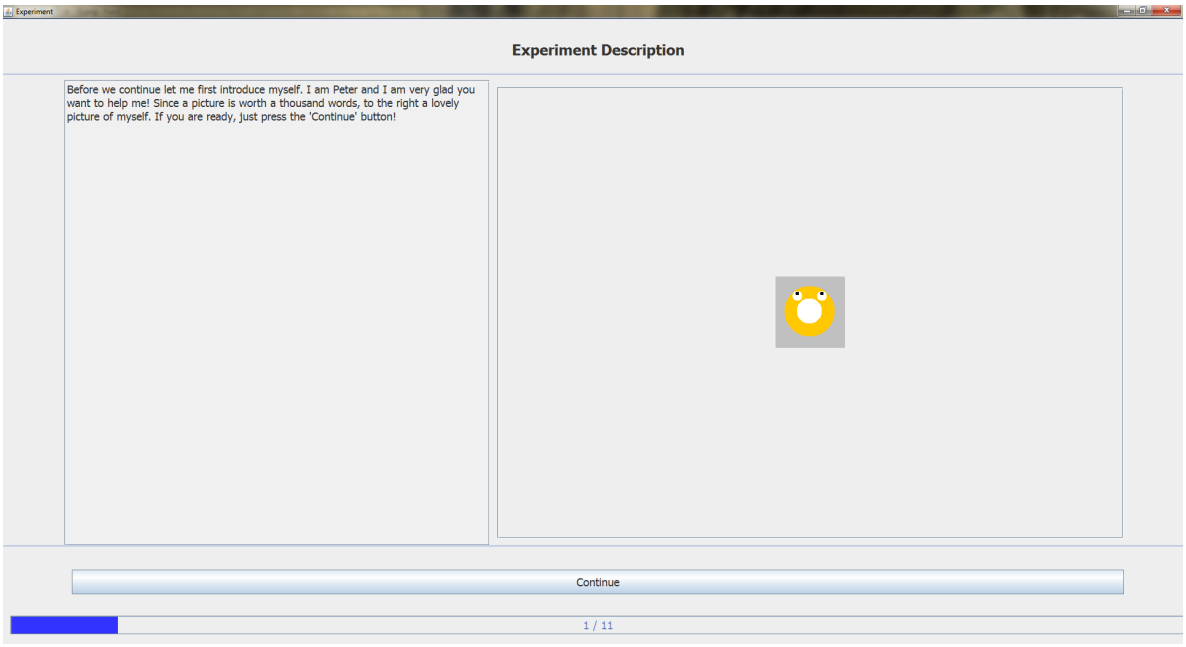


Figure D.2 – First description screen; introduction.

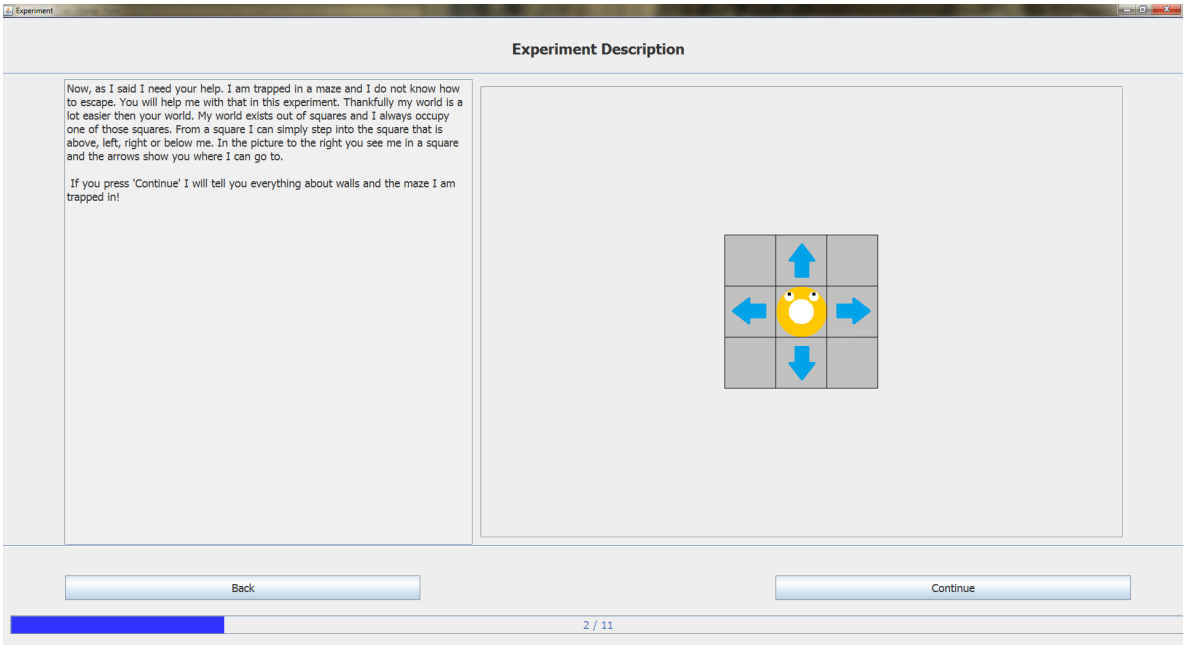


Figure D.3 – Second experiment screen; the actions.

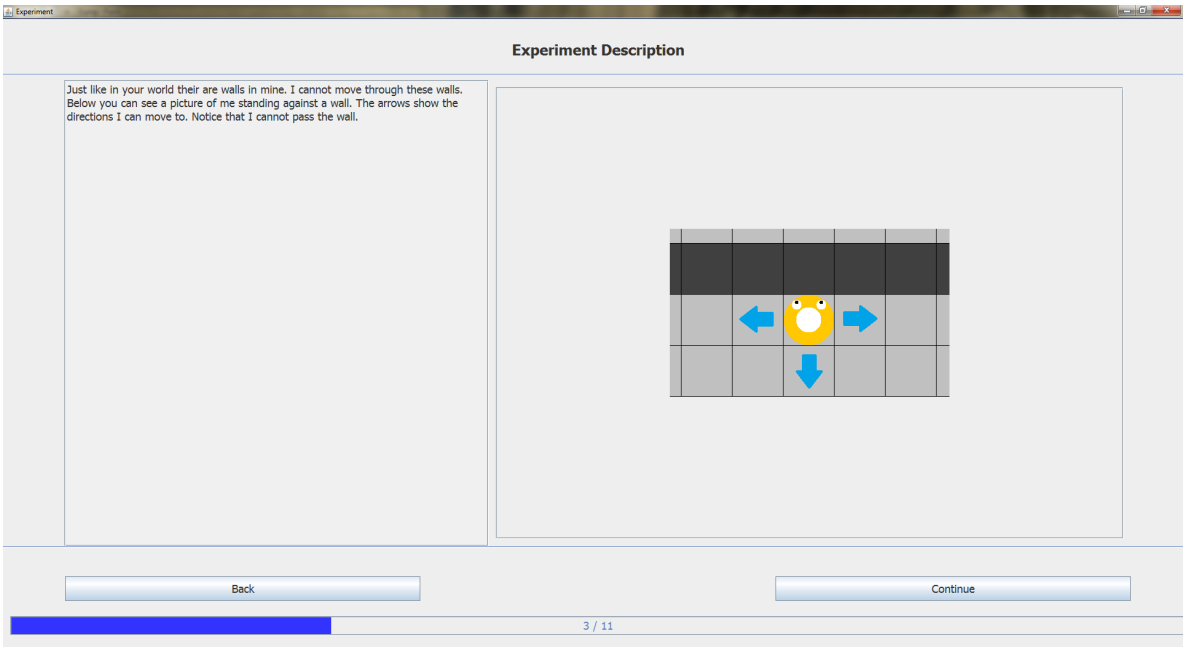


Figure D.4 – Third experiment screen; a wall.

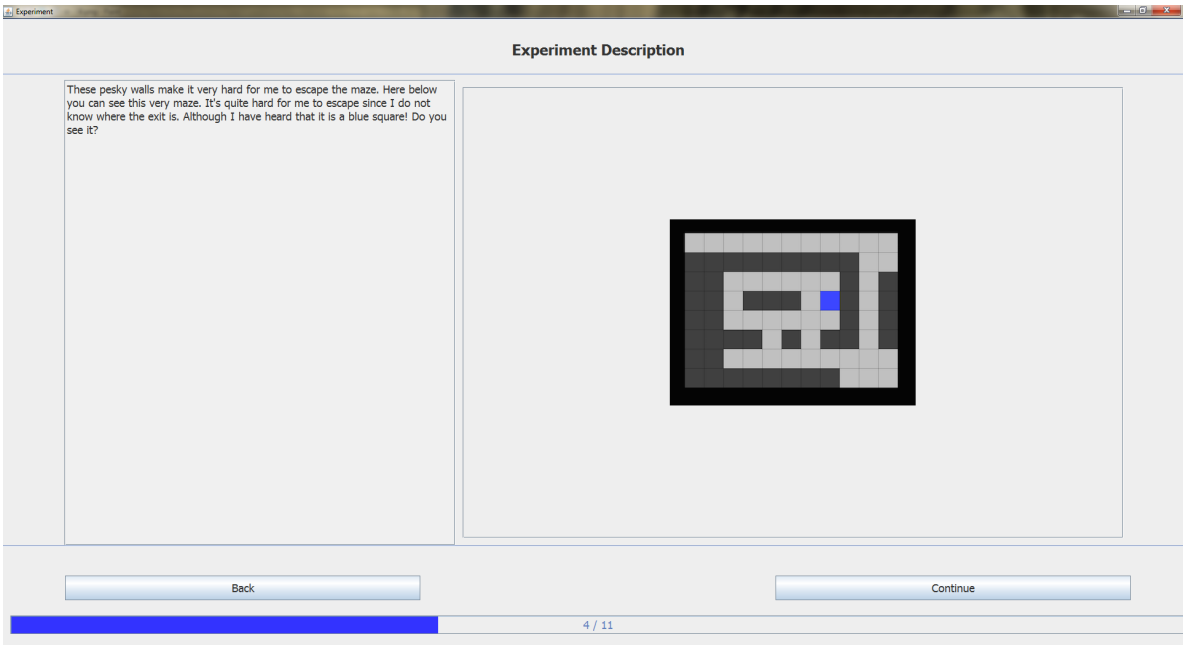


Figure D.5 – Fourth experiment screen; the maze.

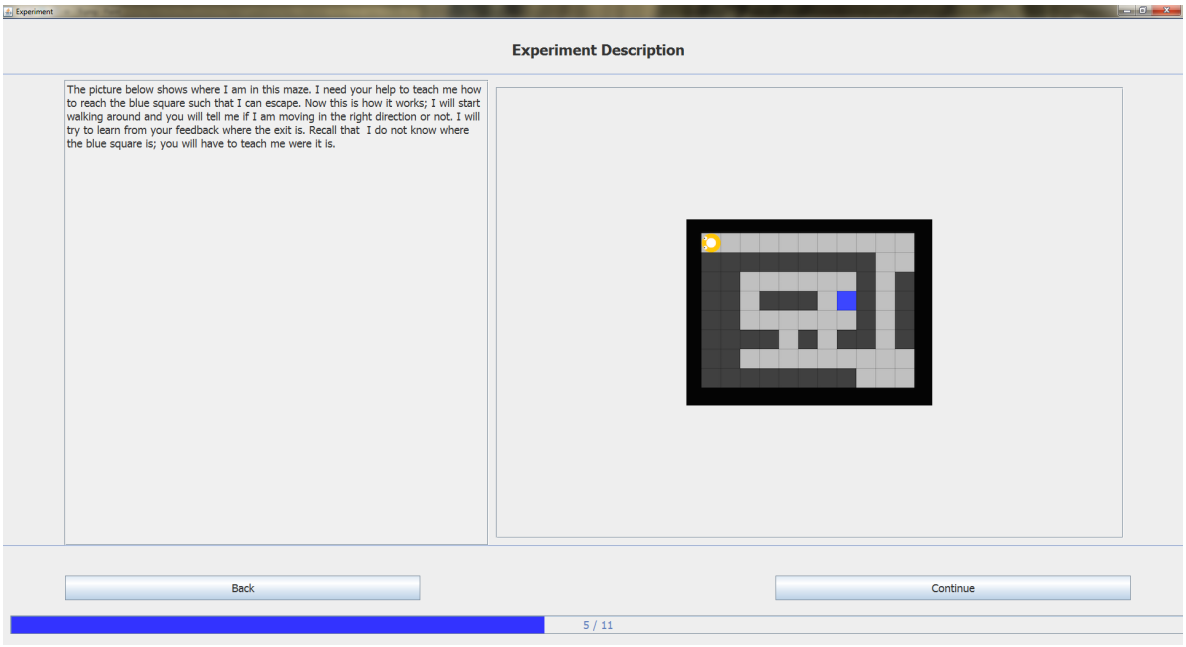


Figure D.6 – Fifth experiment screen; the start position and goal position.

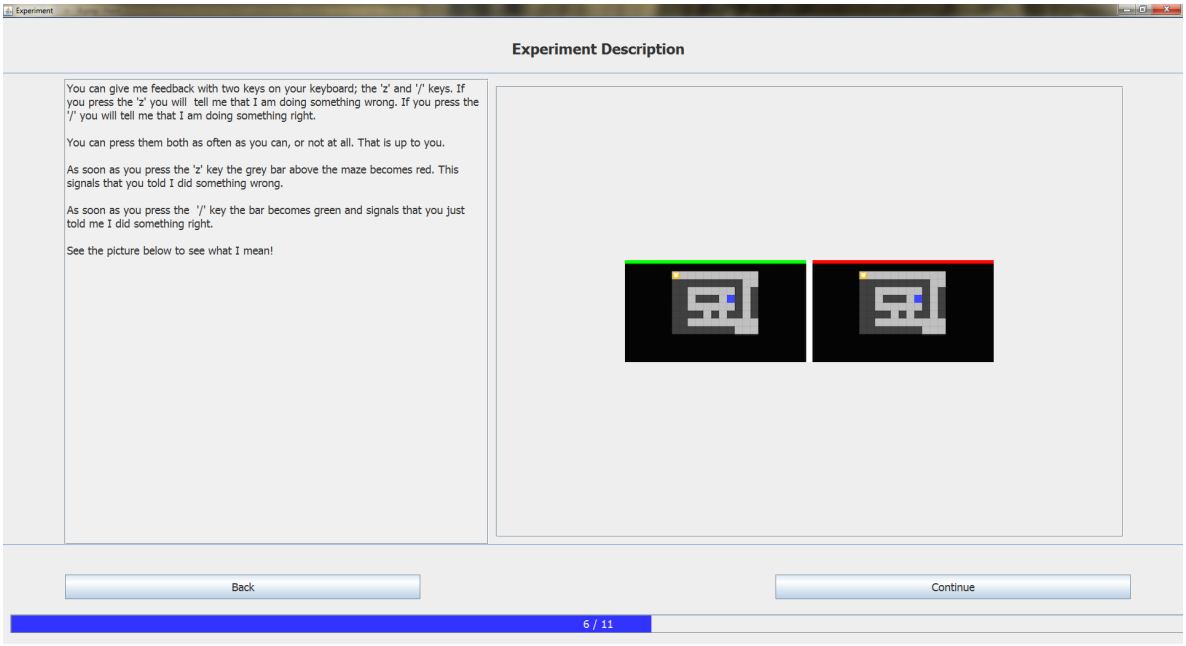


Figure D.7 – Sixth experiment screen; color feedback based on whether the user gave positive/negative feedback.

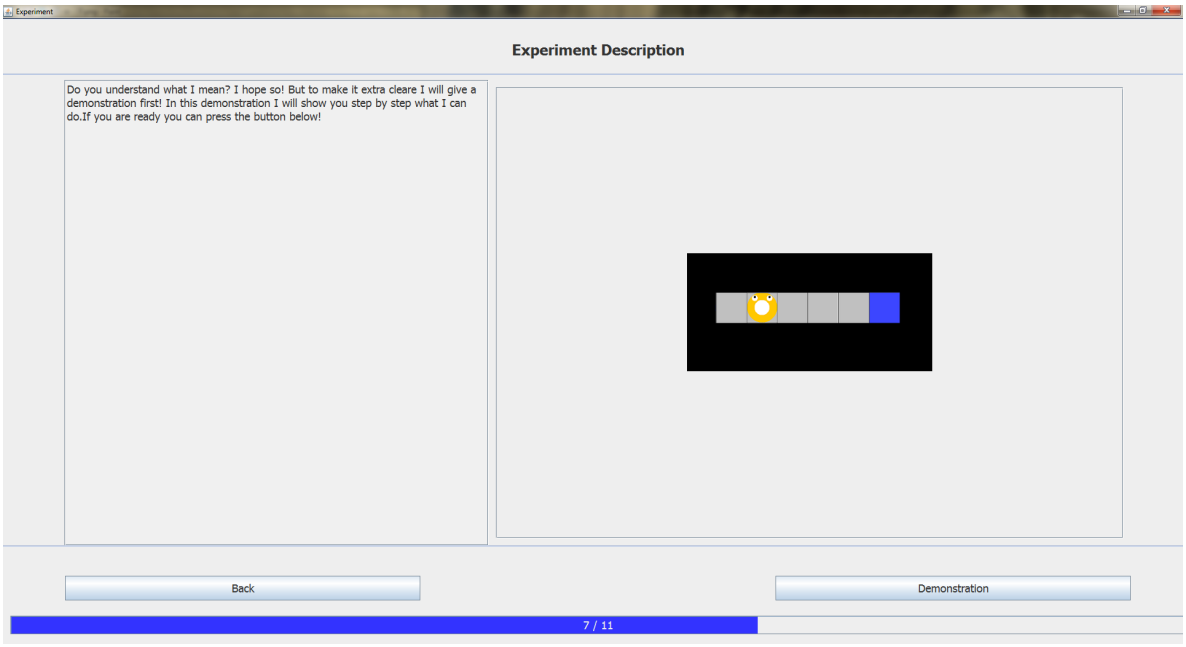


Figure D.8 – Seventh experiment screen; the demonstration.

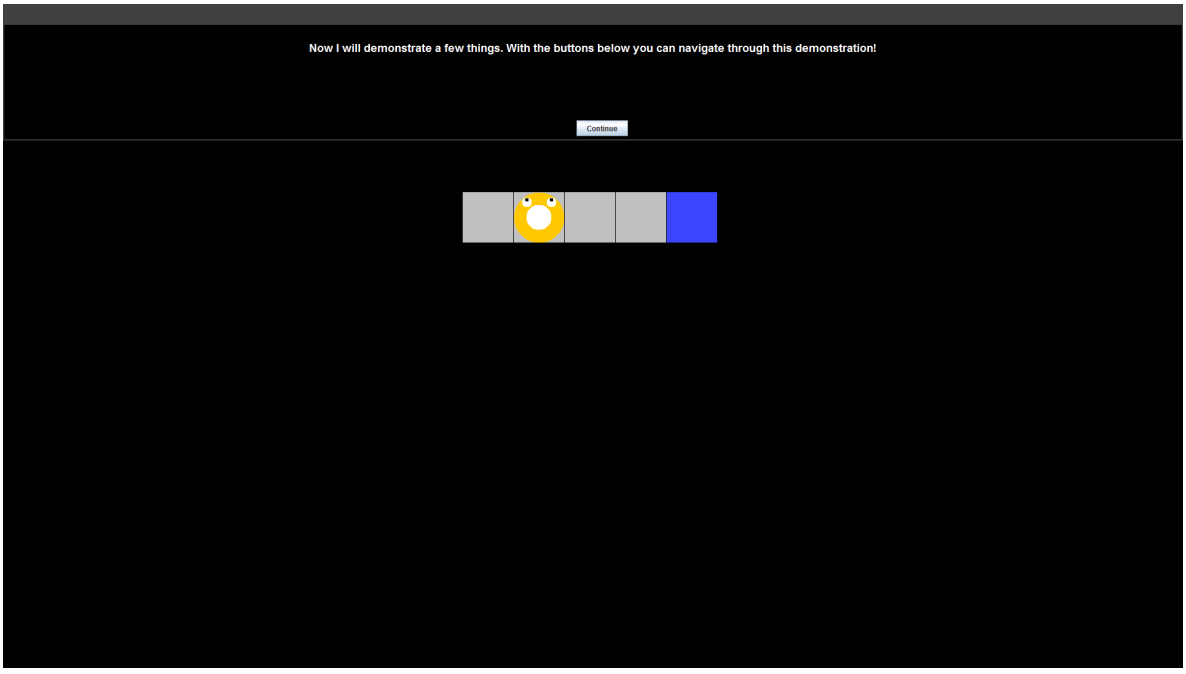


Figure D.9 – First demonstration screen; introduction.

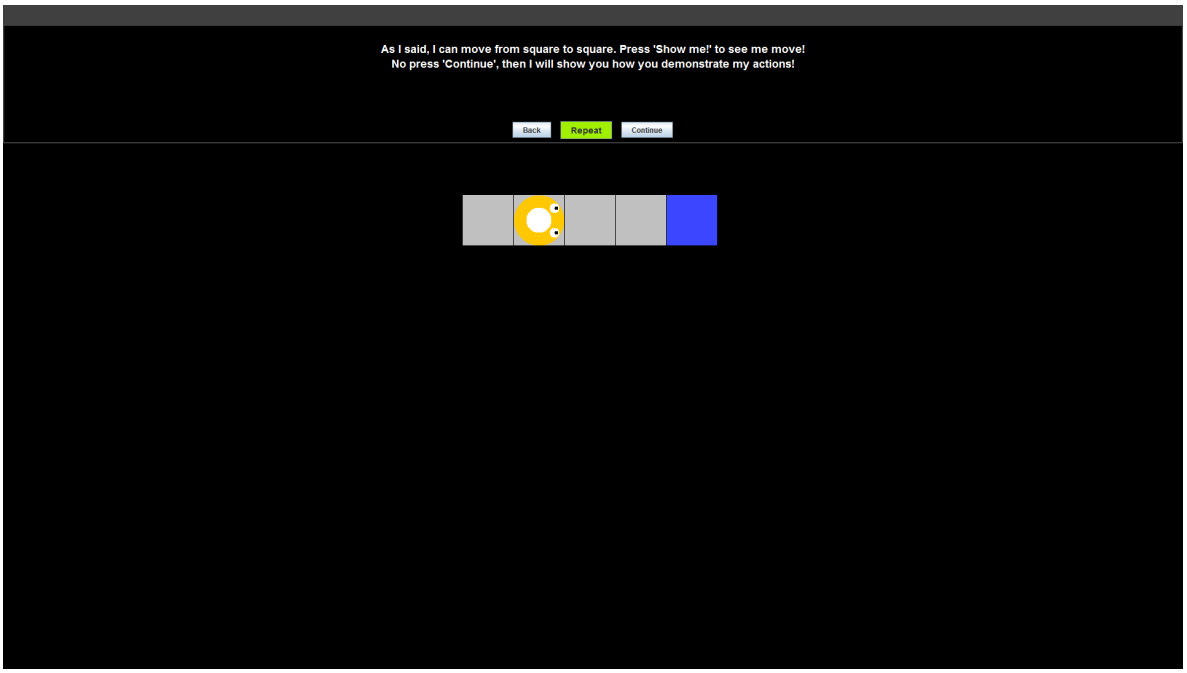


Figure D.10 – Second demonstration screen; example of movement. The screenshot shows the end of each example. The text-lines were shown in appriopriate order after the stated task was completed.

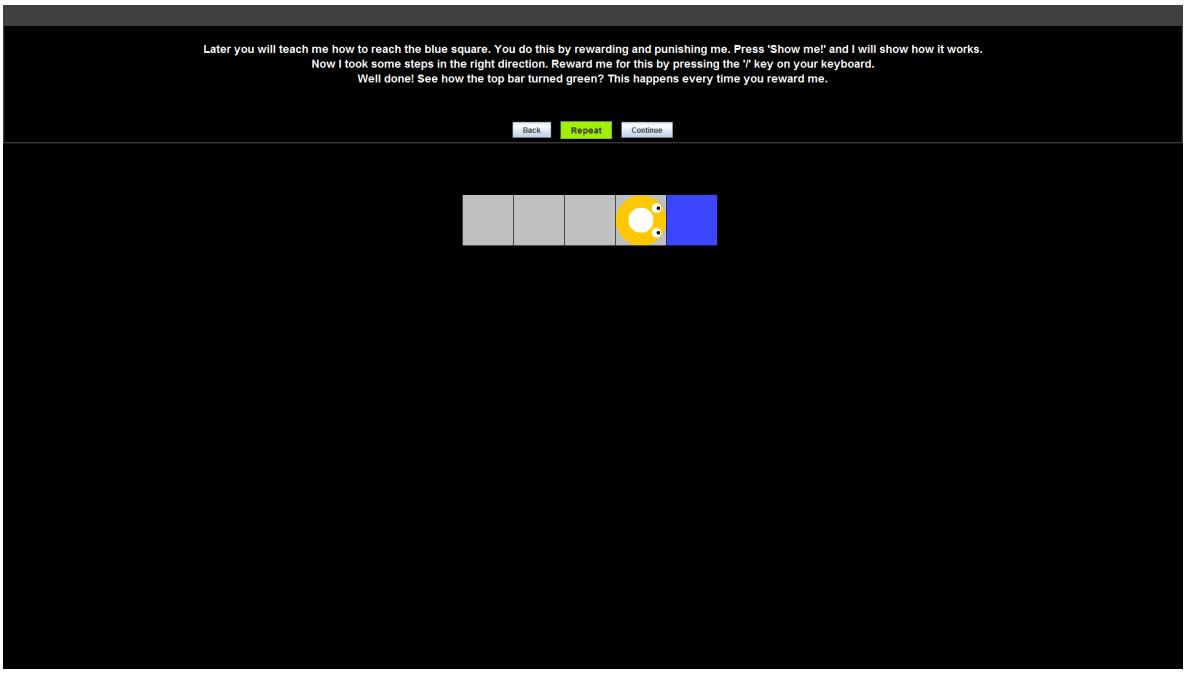


Figure D.11 – Third demonstration screen; example of giving positive feedback.

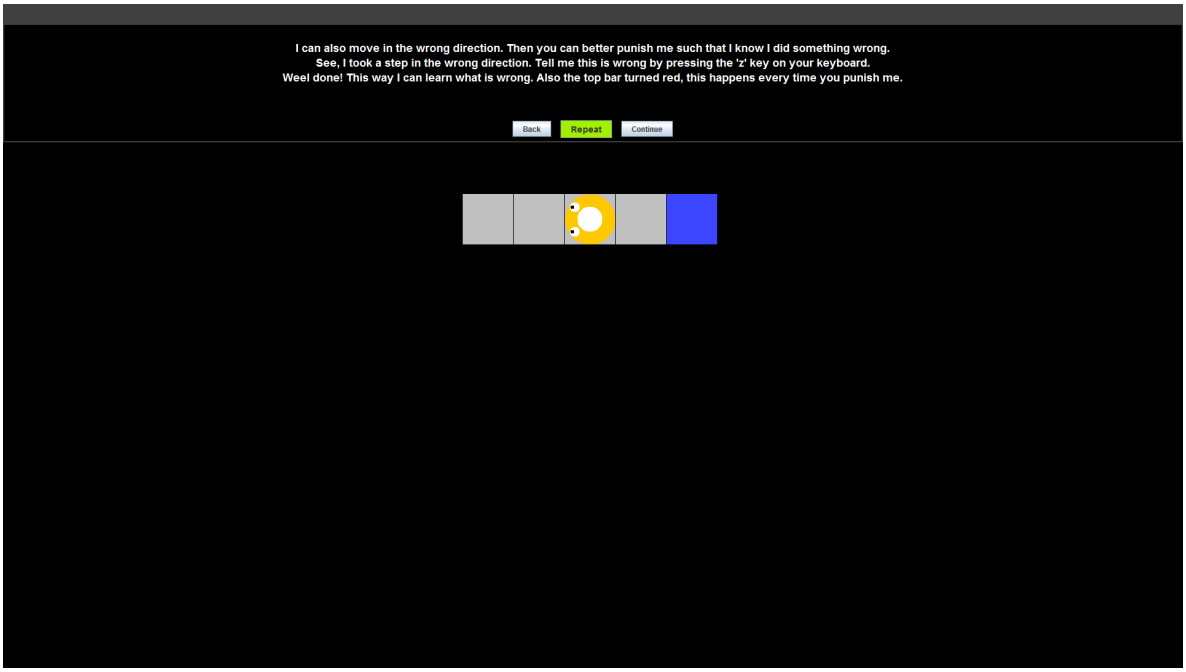


Figure D.12 – Fourth demonstration screen; example of giving negative feedback.

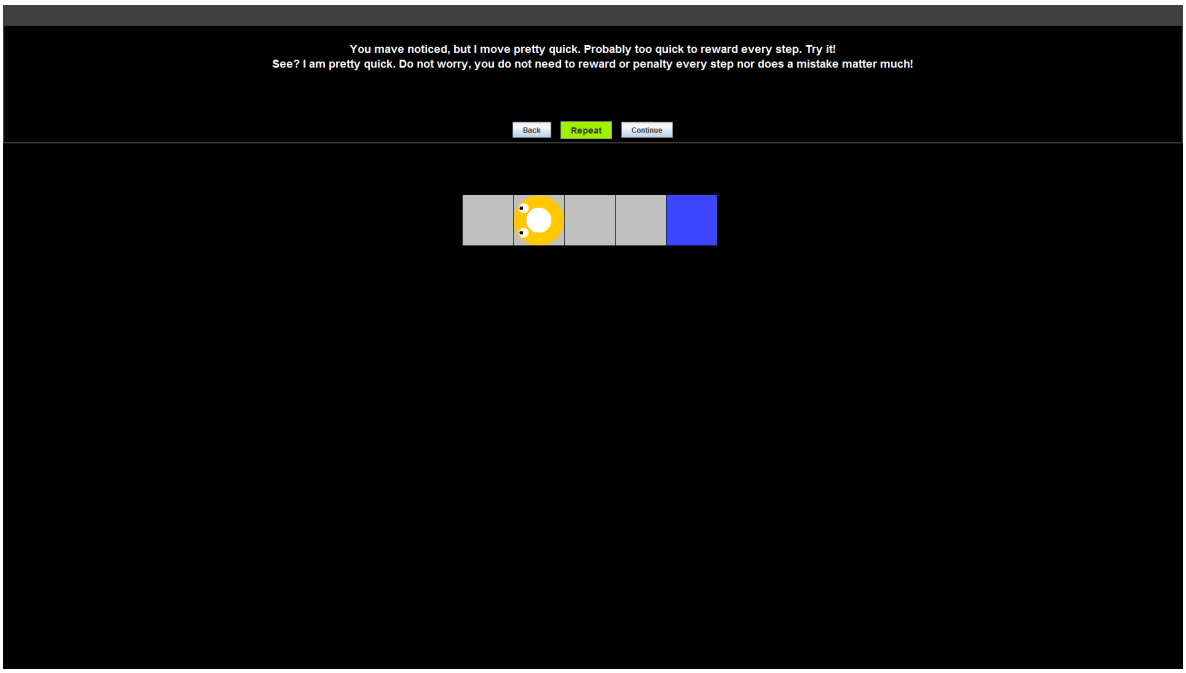


Figure D.13 – Fifth demonstration screen; demonstration of the agent's fast movement.

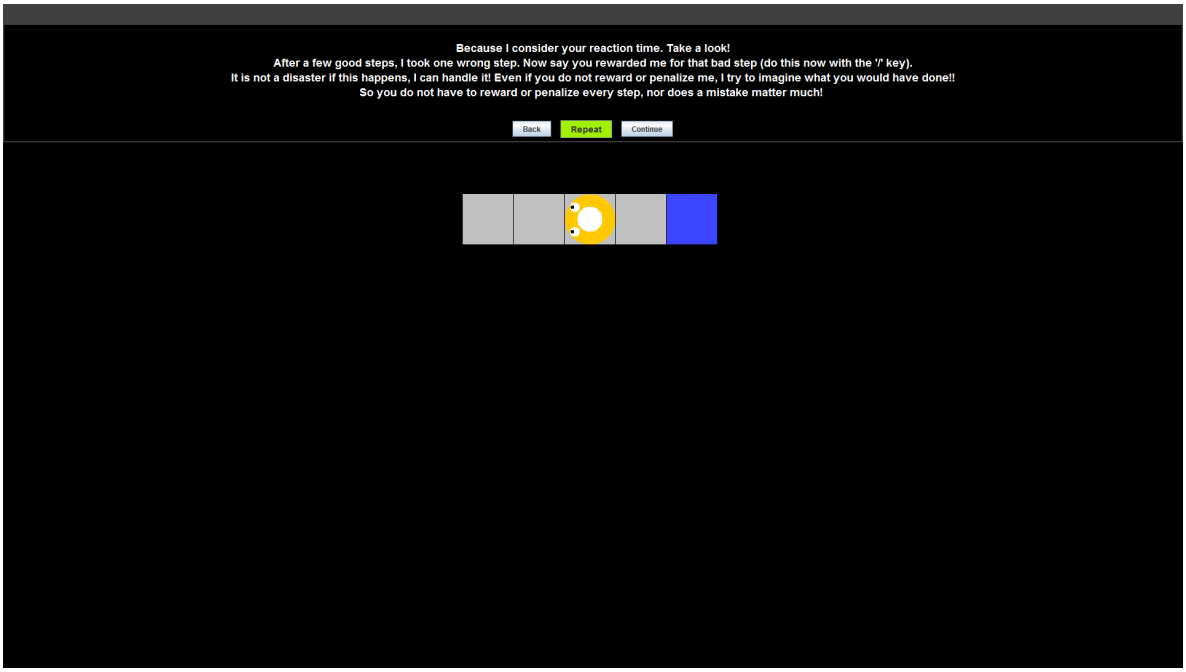


Figure D.14 – Sixth demonstration screen; demonstration of the agent's feedback distribution.

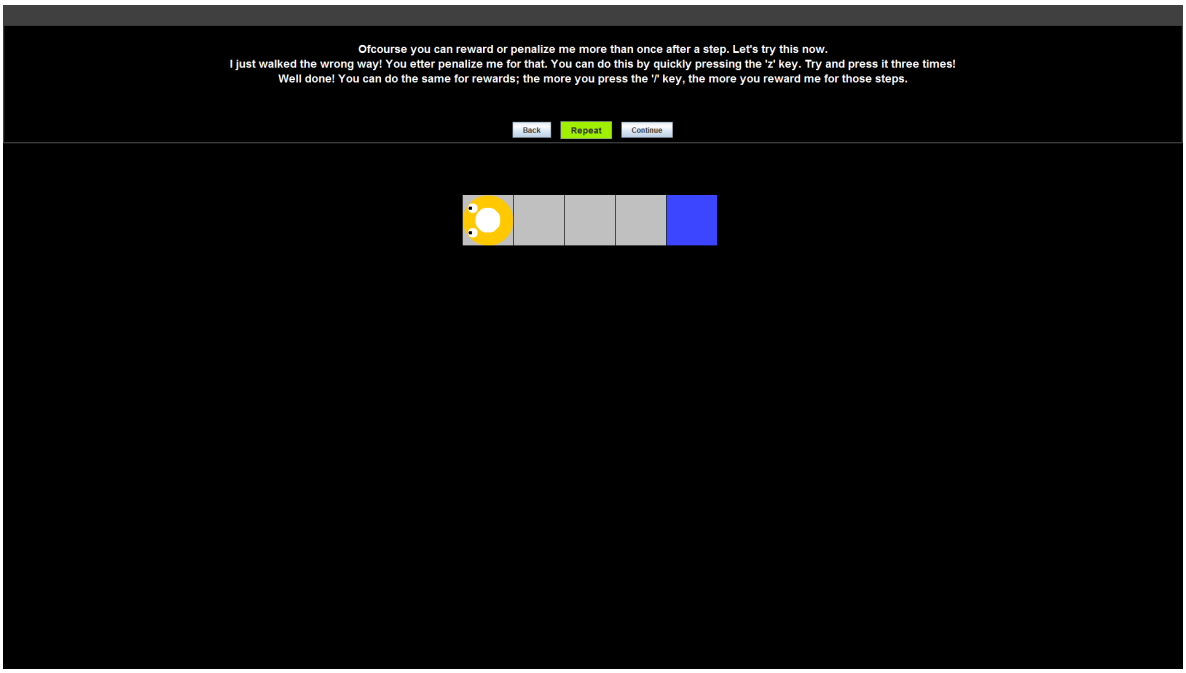


Figure D.15 – Second demonstration screen; example of giving multiple feedback signals.

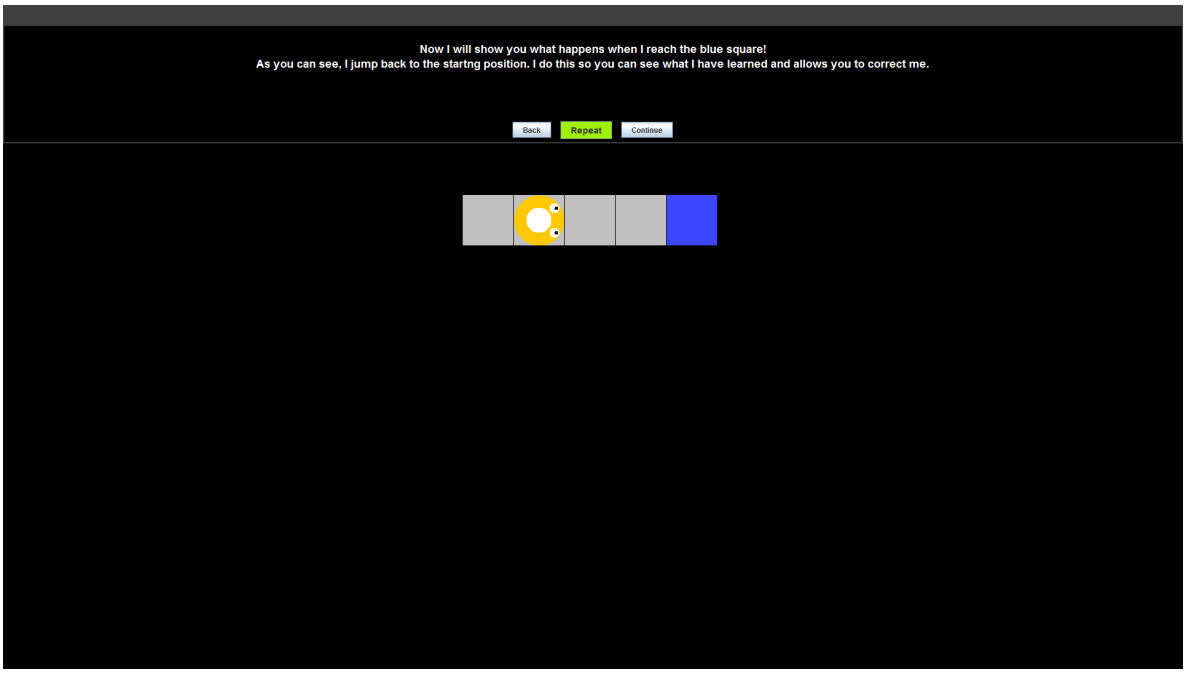


Figure D.16 – Seventh demonstration screen; demonstration of what happens when the blue square (the goal) was reached.

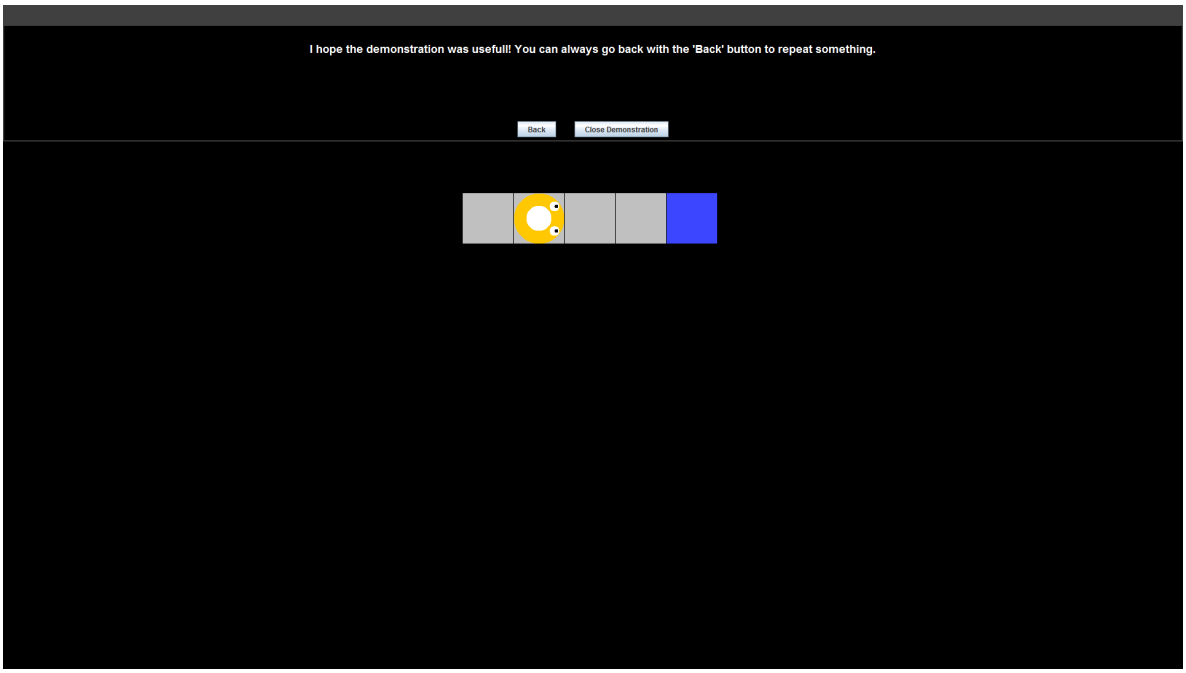


Figure D.17 – Eigth demonstration screen; end of demonstration.

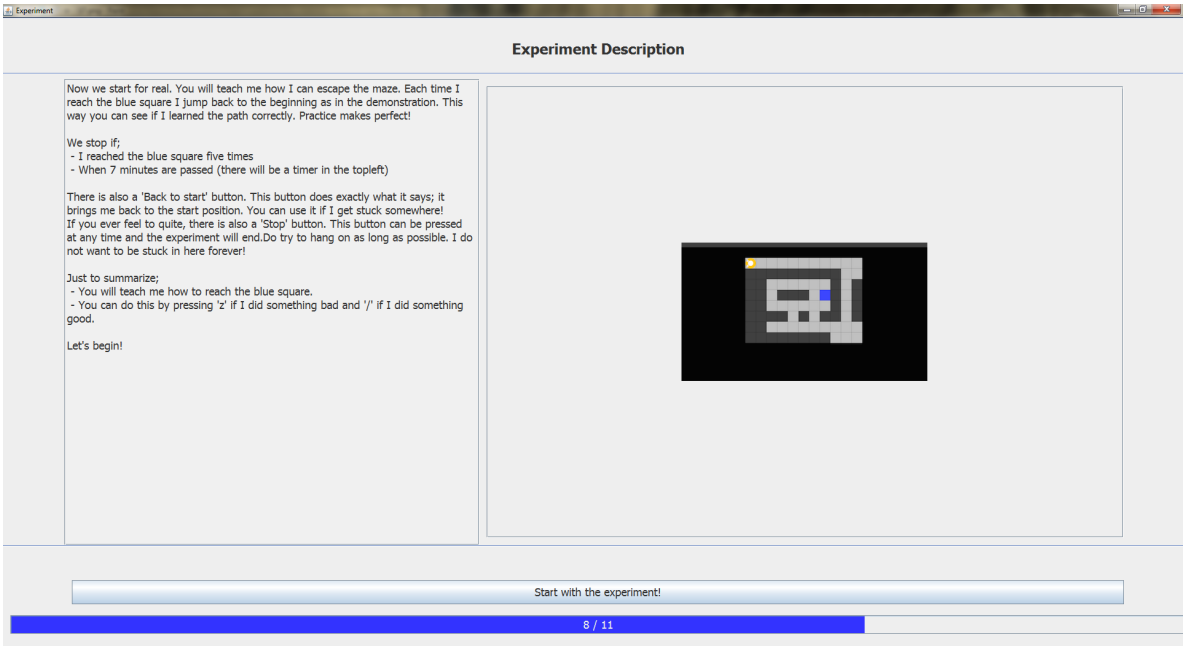


Figure D.18 – Eighth experiment screen; recap.

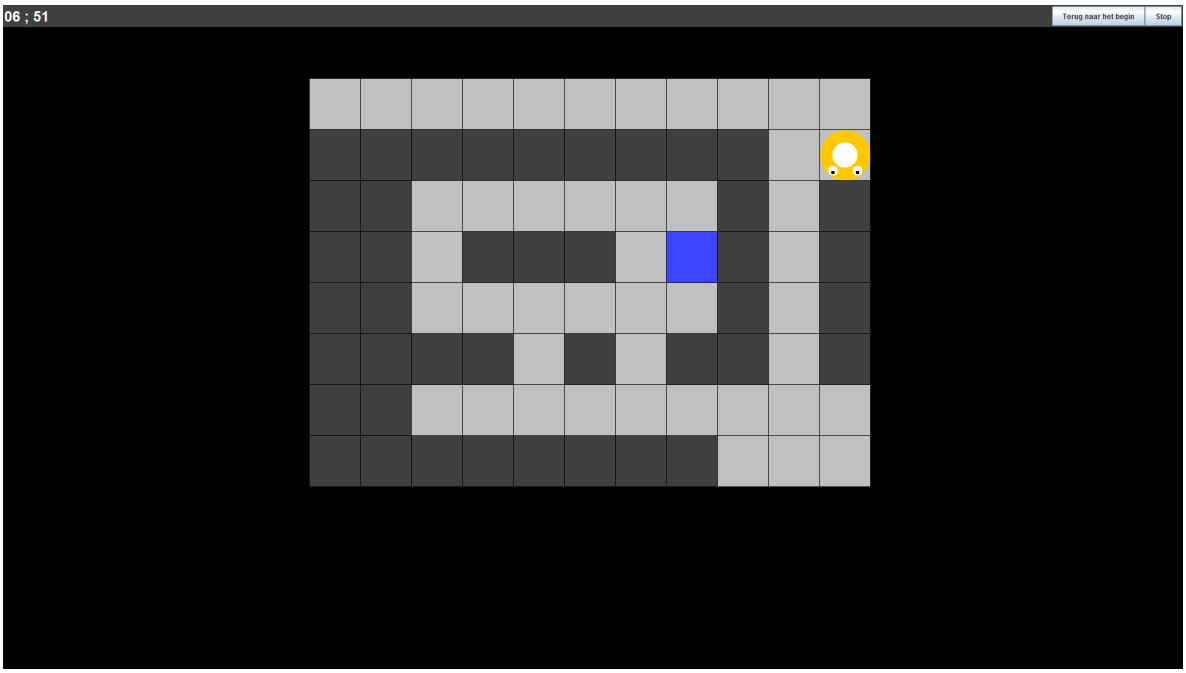


Figure D.19 – The task itself.

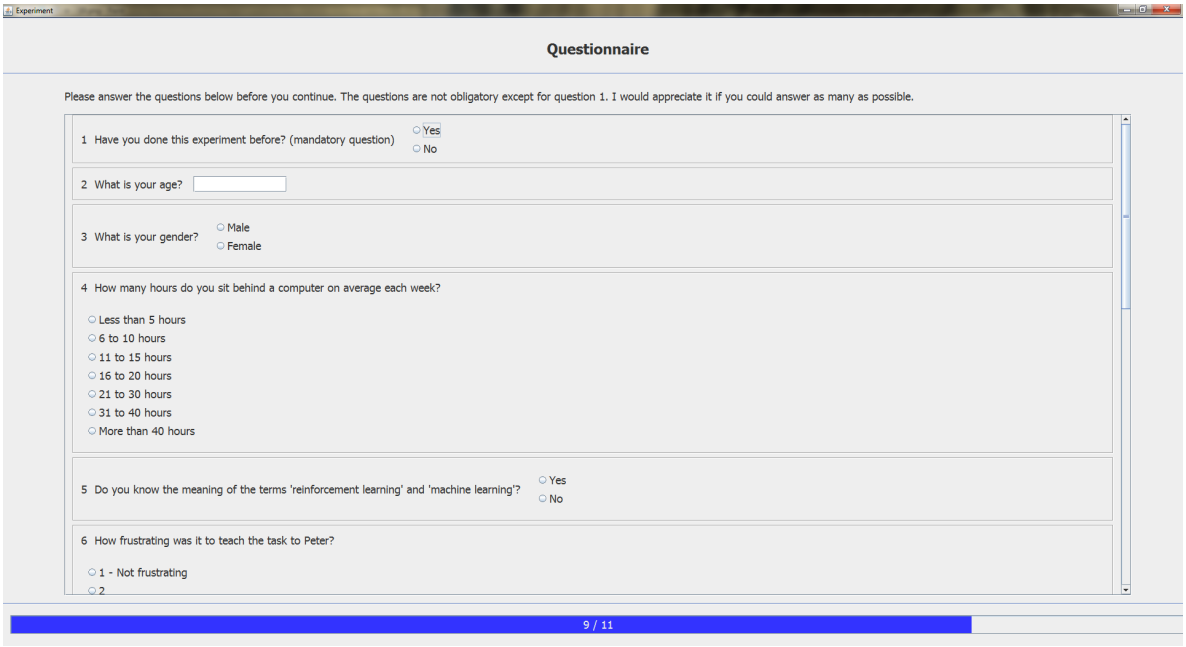


Figure D.20 – Nineth experiment screen; First part of questionnaire.

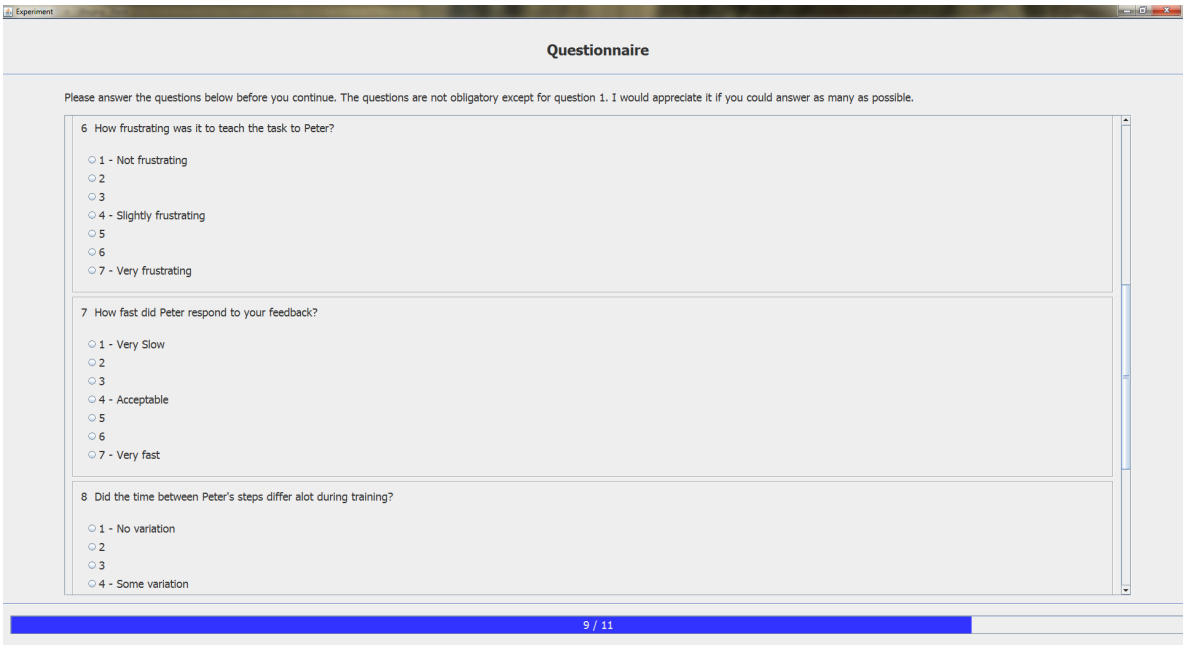


Figure D.21 – Nineth experiment screen; Second part of questionnaire.

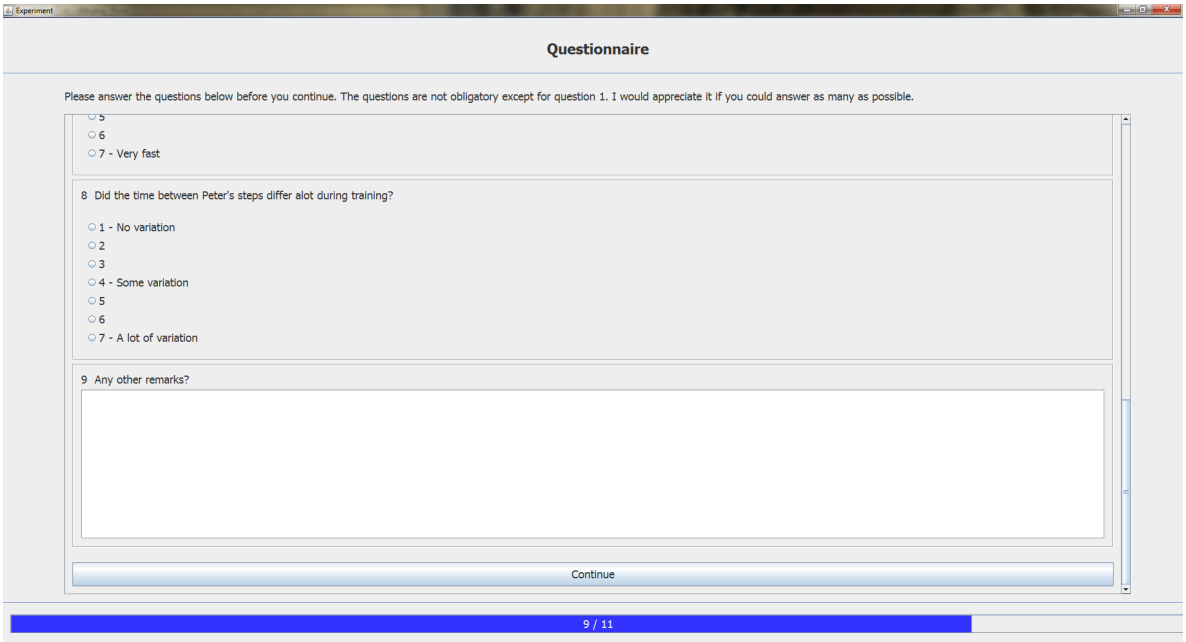


Figure D.22 – Nineth experiment screen; Third part of questionnaire.

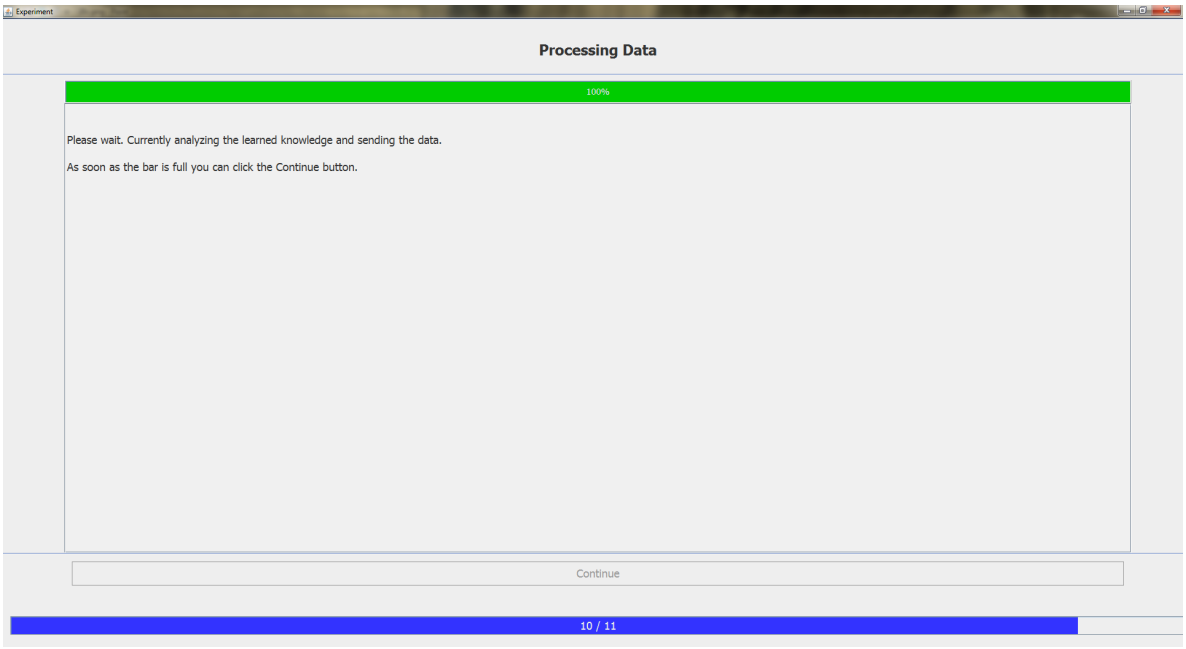


Figure D.23 – Tenth experiment screen; Processing and sending the data.

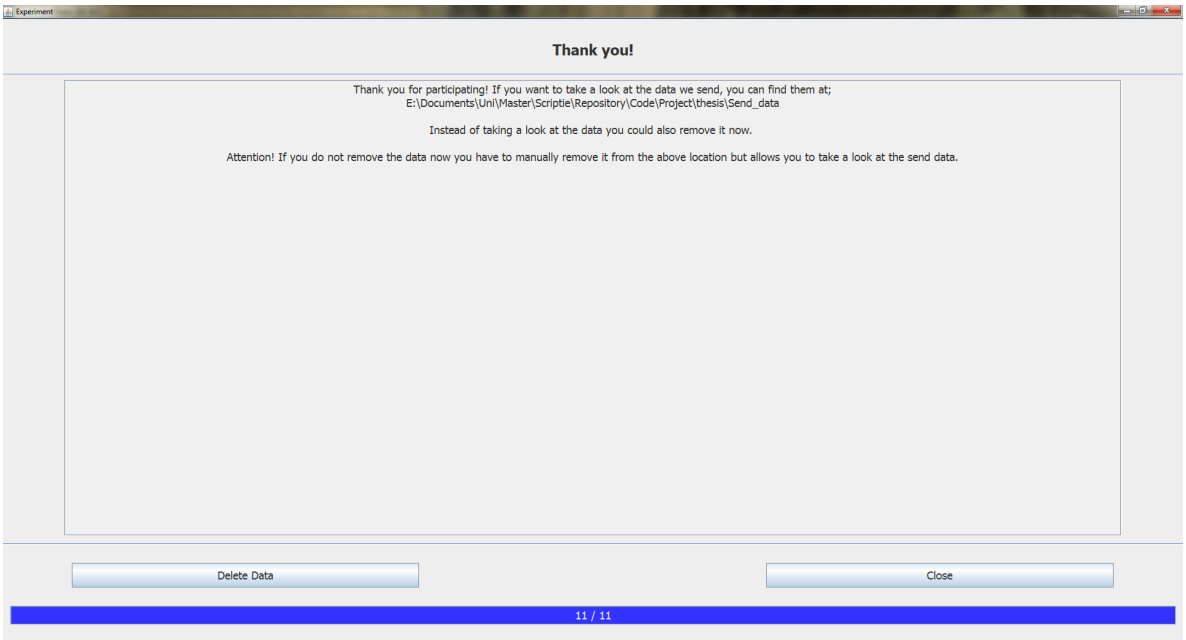


Figure D.24 – Eleventh experiment screen; Final words.

Appendix E

Raw data

This appendix provides the raw data used for the four measurements in the study.

E.1 Number of times the goal was reached

Participant	Baseline	Preventive	Symptom	Combined
1	5	5	5	5
2	1	5	1	5
3	0	5	5	5
4	1	0	3	5
5	0	5	5	5
6	0	5	5	5
7	1	1	5	5
8	5	2	5	5
9	0	5	0	5
10	0	0	5	5
11	0	2	5	5
12	0	5	5	5
13	0	5	5	5
14	5	5	3	5
15	5	5	4	5

Table E.1 – The number of times the goal was reached by each participant's agent of every condition.

E.2 Number of actions per consequetively reached goal

Participant	First Goal			
	Baseline	Preventive	Symptom	Combined
1	244	58	136	114
2	108	68	116	24
3	nan	100	76	196
4	114	nan	82	138
5	nan	54	66	86
6	nan	124	110	62
7	250	298	114	92
8	144	74	58	248
9	nan	202	nan	108
10	nan	nan	78	120
11	nan	66	96	62
12	nan	88	132	122
13	nan	56	124	60
14	218	344	128	84
15	144	108	128	66

Table E.2 – The number of actions the agents of each participant in every condition needed to reach the goal for the first time.

Participant	Second Goal			
	Baseline	Preventive	Symptom	Combined
1	57	45	39	83
2	nan	27	nan	83
3	nan	29	45	109
4	nan	nan	77	131
5	nan	85	95	63
6	nan	41	111	47
7	nan	nan	41	123
8	33	65	57	87
9	nan	55	nan	53
10	nan	nan	123	85
11	nan	81	45	73
12	nan	127	63	81
13	nan	37	179	150
14	37	35	55	87
15	29	33	221	45

Table E.3 – The number of actions the agents of each participant in every condition needed to reach the goal for the second time.

Participant	Fourth			
	Baseline	Preventive	Symptom	Combined
1	35	27	35	35
2	nan	25	nan	31
3	nan	45	35	125
4	nan	nan	97	39
5	nan	29	25	33
6	nan	25	39	51
7	nan	nan	33	35
8	35	nan	27	45
9	nan	29	nan	25
10	nan	nan	61	45
11	nan	nan	27	41
12	nan	143	65	29
13	nan	33	59	41
14	25	45	45	43
15	93	28	49	37

Table E.4 – The number of actions the agents of each participant in every condition needed to reach the goal third time.

Participant	Fourth Goal			
	Baseline	Preventive	Symptom	Combined
1	35	29	61	55
2	nan	25	nan	31
3	nan	31	29	29
4	nan	nan	nan	33
5	nan	29	25	29
6	nan	29	41	37
7	nan	nan	23	29
8	29	nan	43	55
9	nan	27	nan	31
10	nan	nan	25	167
11	nan	nan	27	35
12	nan	29	53	23
13	nan	25	29	73
14	27	43	nan	33
15	27	27	113	25

Table E.5 – The number of actions the agents of each participant in every condition needed to reach the goal for the fourth time.

Participant	Fifth Goal			
	Baseline	Preventive	Symptom	Combined
1	31	35	51	55
2	nan	27	nan	31
3	nan	39	27	23
4	nan	nan	nan	41
5	nan	29	25	33
6	nan	55	27	35
7	nan	nan	25	31
8	31	nan	49	71
9	nan	27	nan	33
10	nan	nan	33	51
11	nan	nan	27	43
12	nan	27	25	27
13	nan	25	51	25
14	27	55	nan	27
15	23	27	nan	23

Table E.6 – The number of actions the agents of each participant in every condition needed to reach the goal for fifth time.

E.3 Number states experienced while repeating a behavior

Participant	Baseline	Preventive	Symptom	Combined
1	99	24	77	81
2	156	21	159	46
3	109	52	49	130
4	158	165	155	100
5	154	43	53	57
6	171	55	79	55
7	163	167	54	76
8	59	36	41	132
9	86	77	0	58
10	152	28	81	133
11	176	43	52	60
12	167	106	79	64
13	174	21	113	81
14	78	142	145	63
15	81	39	157	40

Table E.7 – The number of states each participant's agent experienced while repeating some behavior for every condition.

E.4 Number actions required to reach the goal from every maze position

E.4.1 Training maze

Position		Baseline; Participants														
x	y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	28	nan	nan	nan	nan	nan	nan	30	nan	nan	nan	nan	nan	24	22
1	0	25	nan	nan	nan	nan	nan	nan	33	nan	nan	nan	nan	nan	23	21
2	0	24	nan	nan	nan	nan	nan	nan	26	nan	nan	nan	nan	nan	26	22
3	0	23	nan	nan	nan	nan	nan	nan	29	nan	nan	nan	nan	nan	19	19
4	0	22	nan	nan	nan	nan	nan	nan	26	nan	nan	nan	nan	nan	18	24
5	0	29	nan	nan	nan	nan	nan	nan	21	nan	nan	nan	nan	nan	17	17
6	0	20	nan	nan	nan	nan	nan	nan	24	nan	nan	nan	nan	nan	28	16
7	0	19	nan	nan	nan	nan	nan	nan	21	nan	nan	nan	nan	103	17	15
8	0	18	nan	nan	nan	nan	nan	nan	24	296	nan	nan	nan	nan	22	16
9	0	21	nan	nan	nan	nan	nan	nan	25	nan	nan	nan	nan	nan	13	13
10	0	18	nan	nan	nan	nan	nan	nan	22	nan	nan	nan	nan	96	16	16
9	1	81	nan	nan	nan	nan	nan	nan	16	nan	nan	nan	nan	nan	12	12
10	1	25	nan	nan	nan	nan	nan	nan	19	83	nan	nan	nan	nan	19	13
2	2	10	36	16	nan	128	60	nan	16	26	nan	62	20	54	12	14
3	2	5	19	213	nan	67	33	nan	5	33	19	35	15	5	15	15
4	2	4	20	28	nan	42	58	16	4	104	54	72	20	134	6	16
5	2	3	47	35	nan	nan	9	33	3	3	7	11	17	41	17	5
6	2	4	10	14	nan	4	nan	nan	2	6	4	46	2	196	2	4
7	2	1	1	1	1	41	1	7	1	1	1	1	1	1	1	1
9	2	17	nan	nan	nan	nan	nan	nan	19	nan	nan	nan	nan	43	13	15
2	3	13	23	17	nan	45	71	nan	7	135	nan	81	51	15	7	7
6	3	1	7	21	nan	3	1	1	1	1	7	1	1	7	1	3
9	3	26	nan	nan	nan	nan	80	nan	14	nan	nan	nan	nan	128	16	12
2	4	6	10	26	nan	16	26	nan	24	70	58	10	36	86	8	22
3	4	5	5	nan	nan	19	81	nan	5	15	nan	23	135	83	5	9
4	4	4	8	136	nan	32	8	nan	4	18	80	32	46	26	4	4
5	4	3	17	nan	nan	nan	21	nan	3	69	nan	3	15	29	7	9
6	4	2	2	2	nan	118	162	nan	2	46	4	10	10	6	2	2
7	4	3	1	1	nan	1	1	1	1	1	21	1	35	1	3	1
9	4	13	nan	nan	nan	41	121	nan	13	nan	nan	nan	nan	141	9	9
4	5	7	49	nan	nan	9	13	nan	5	31	nan	nan	51	49	7	5
6	5	3	29	nan	nan	371	5	nan	3	nan	23	75	49	5	3	3
9	5	14	nan	nan	nan	36	90	nan	12	140	nan	nan	60	nan	8	8
2	6	14	38	100	nan	20	142	nan	8	116	nan	106	298	80	14	14
3	6	7	25	nan	nan	nan	37	nan	9	79	nan	93	35	29	9	25
4	6	6	12	18	nan	66	176	nan	8	nan	nan	nan	170	132	12	10
5	6	11	9	nan	nan	19	49	nan	7	17	nan	11	103	47	7	5
6	6	8	14	nan	nan	nan	40	nan	10	nan	nan	48	68	62	4	4
7	6	9	5	nan	nan	nan	109	nan	11	77	nan	nan	101	169	5	7
8	6	16	24	nan	nan	84	246	nan	10	72	nan	18	336	248	8	10
9	6	11	21	nan	nan	53	245	nan	11	nan	nan	nan	149	71	7	17
10	6	18	nan	nan	nan	nan	nan	nan	12	nan	nan	30	66	78	12	8
8	7	11	11	nan	nan	nan	295	nan	21	nan	nan	165	91	nan	7	13
9	7	12	nan	nan	nan	nan	122	nan	14	nan	nan	74	104	318	18	14
10	7	15	nan	nan	nan	nan	267	nan	17	395	nan	43	459	117	9	13

Table E.8 – Number actions it took for all baseline agents to reach the goal from every maze position. All “nan"

Position		Preventive; Participants														
x	y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	26	24	nan	nan	22	nan	nan	nan	nan	nan	nan	24	nan	24	26
1	0	25	23	nan	nan	21	nan	nan	nan	nan	nan	nan	23	nan	23	25
2	0	24	22	nan	nan	20	nan	nan	nan	nan	nan	nan	22	nan	22	24
3	0	23	21	nan	nan	19	nan	nan	nan	nan	nan	nan	21	nan	21	23
4	0	22	20	nan	nan	18	nan	nan	nan	nan	nan	nan	20	nan	20	22
5	0	21	19	nan	nan	17	nan	nan	nan	nan	nan	nan	19	nan	19	21
6	0	20	18	nan	nan	16	nan	nan	nan	nan	nan	nan	18	nan	18	20
7	0	19	17	nan	nan	15	nan	nan	nan	nan	nan	nan	17	nan	17	19
8	0	18	16	nan	nan	14	nan	nan	nan	nan	nan	nan	16	nan	16	18
9	0	17	15	nan	nan	13	nan	nan	nan	nan	nan	nan	15	nan	15	17
10	0	18	nan	nan	nan	14	nan	nan	nan	nan	nan	nan	nan	nan	16	nan
9	1	16	14	nan	nan	12	nan	nan	nan	nan	nan	nan	14	nan	14	16
10	1	17	nan	nan	nan	13	nan	nan	nan	nan	nan	nan	nan	nan	15	nan
2	2	6	6	6	6	6	6	nan	6	6	nan	nan	6	6	6	6
3	2	5	5	5	5	5	5	nan	5	5	nan	nan	5	5	5	5
4	2	4	4	4	4	4	4	nan	4	4	nan	4	4	4	4	4
5	2	3	3	3	3	3	3	nan	3	3	nan	3	3	3	3	3
6	2	2	2	2	2	2	2	nan	2	2	nan	2	2	2	2	2
7	2	1	1	1	1	1	1	1	1	1	nan	1	1	1	1	1
9	2	15	13	nan	nan	11	nan	nan	nan	nan	nan	nan	13	nan	13	15
2	3	7	9	nan	nan	nan	nan	nan	nan	7	nan	nan	9	7	9	7
6	3	1	3	1	1	1	1	1	1	nan	nan	1	3	nan	3	1
9	3	14	12	nan	nan	10	nan	nan	nan	nan	nan	nan	12	nan	12	14
2	4	6	8	nan	nan	nan	nan	nan	nan	8	nan	nan	8	8	8	6
3	4	5	7	nan	nan	nan	nan	nan	nan	nan	nan	nan	7	nan	7	5
4	4	4	6	nan	nan	nan	nan	nan	nan	nan	nan	nan	6	nan	6	4
5	4	3	5	nan	nan	3	nan	nan	nan	nan	nan	nan	5	nan	5	3
6	4	2	4	nan	nan	2	nan	nan	nan	nan	nan	nan	4	nan	4	2
7	4	1	1	nan	nan	1	nan	1	1	1	nan	nan	1	1	1	1
9	4	13	11	nan	nan	9	nan	nan	nan	nan	nan	nan	11	nan	11	13
4	5	5	7	nan	nan	nan	nan	nan	nan	nan	nan	nan	7	nan	7	5
6	5	3	5	nan	nan	3	nan	nan	nan	nan	nan	nan	5	nan	5	3
9	5	12	10	nan	nan	8	nan	nan	nan	nan	nan	nan	10	nan	10	12
2	6	8	10	nan	nan	nan	nan	nan	nan	nan	nan	nan	10	nan	10	8
3	6	7	9	nan	nan	nan	nan	nan	nan	nan	nan	nan	9	nan	9	7
4	6	6	8	nan	nan	nan	nan	nan	nan	nan	nan	nan	8	nan	8	6
5	6	7	7	nan	nan	5	nan	nan	nan	nan	nan	nan	7	nan	7	7
6	6	8	6	nan	nan	4	nan	nan	nan	nan	nan	nan	6	nan	6	8
7	6	9	7	nan	nan	5	nan	nan	nan	nan	nan	nan	7	nan	7	9
8	6	10	8	nan	nan	6	nan	nan	nan	nan	nan	nan	8	nan	8	10
9	6	11	9	nan	nan	7	nan	nan	nan	nan	nan	nan	9	nan	9	11
10	6	12	10	nan	nan	8	nan	nan	nan	nan	nan	nan	10	nan	10	12
8	7	11	9	nan	nan	7	nan	nan	nan	nan	nan	nan	9	nan	9	11
9	7	12	10	nan	nan	8	nan	nan	nan	nan	nan	nan	10	nan	10	12
10	7	13	11	nan	nan	9	nan	nan	nan	nan	nan	nan	11	nan	11	13

Table E.9 – Number actions it took for all agents with the preventive solution to reach the goal from every maze position. All “nan" values are positions from which the goal was not reached within 525 actions.

Position		Symptom; Participants														
x	y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	nan	nan	28	nan	22	22	22	nan	270	22	22	22	nan	nan	nan
1	0	nan	nan	21	nan	21	21	21	nan	283	21	21	21	nan	nan	nan
2	0	nan	nan	24	nan	20	20	20	nan	nan	20	20	20	nan	nan	nan
3	0	nan	nan	19	nan	19	19	19	nan	453	19	19	19	nan	nan	nan
4	0	nan	nan	22	nan	18	18	18	nan	214	18	18	18	nan	nan	nan
5	0	nan	nan	17	nan	17	17	17	nan	nan	17	17	17	nan	nan	nan
6	0	nan	nan	18	nan	16	16	16	nan	nan	16	16	16	nan	nan	nan
7	0	nan	nan	15	nan	15	15	15	nan	375	15	15	15	nan	nan	nan
8	0	nan	nan	16	nan	14	14	14	nan	164	14	14	14	nan	nan	nan
9	0	nan	nan	17	nan	13	13	13	nan	nan	13	13	13	nan	nan	nan
10	0	nan	nan	16	nan	14	16	14	nan	nan	26	18	14	nan	nan	nan
9	1	nan	nan	14	nan	12	12	12	nan	nan	12	12	12	nan	nan	nan
10	1	nan	nan	21	nan	13	15	15	nan	431	13	23	13	nan	nan	nan
2	2	6	6	6	nan	6	12	10	8	84	8	14	6	nan	50	nan
3	2	5	13	5	nan	5	15	17	13	55	9	5	5	nan	11	nan
4	2	4	10	4	nan	4	4	6	6	32	14	4	4	nan	4	nan
5	2	3	5	3	nan	3	3	3	3	41	9	5	3	nan	9	nan
6	2	2	8	2	nan	2	2	2	2	12	2	2	2	nan	16	nan
7	2	1	1	1	1	1	1	3	1	1	3	1	1	1	1	1
9	2	nan	nan	13	nan	11	11	11	nan	nan	11	11	11	nan	nan	nan
2	3	7	11	7	nan	7	53	7	9	77	19	9	9	nan	21	nan
6	3	1	1	1	nan	1	1	1	1	9	1	1	1	nan	1	nan
9	3	nan	nan	14	nan	10	10	10	nan	nan	10	10	10	nan	nan	nan
2	4	12	16	6	nan	22	20	6	8	196	8	6	8	nan	14	nan
3	4	7	7	5	nan	7	5	5	9	21	19	5	11	nan	41	nan
4	4	6	10	4	nan	12	6	4	4	nan	4	4	4	nan	28	nan
5	4	5	7	3	nan	3	5	3	3	7	3	3	3	nan	3	nan
6	4	2	6	2	nan	2	2	2	2	4	2	2	2	nan	2	nan
7	4	1	1	1	1	1	1	1	1	7	3	1	1	nan	1	1
9	4	nan	nan	9	nan	9	9	9	nan	213	9	9	9	nan	nan	nan
4	5	13	11	5	nan	7	7	7	5	nan	9	nan	7	nan	5	nan
6	5	3	3	3	nan	3	3	3	3	nan	3	3	3	nan	3	nan
9	5	nan	nan	8	nan	8	8	8	nan	174	8	8	8	nan	nan	nan
2	6	18	10	16	nan	8	nan	8	8	267	10	nan	8	nan	8	nan
3	6	11	9	13	nan	7	nan	7	11	27	9	nan	7	nan	9	nan
4	6	6	10	8	nan	6	nan	12	8	12	6	nan	6	nan	10	nan
5	6	5	7	5	nan	5	5	5	5	67	7	nan	5	nan	5	nan
6	6	4	4	4	nan	4	4	4	4	12	4	4	4	nan	4	nan
7	6	nan	nan	7	nan	5	5	5	5	31	5	5	5	nan	5	nan
8	6	nan	nan	8	nan	6	6	6	6	58	6	6	6	nan	6	nan
9	6	nan	nan	7	nan	7	7	7	7	225	7	7	7	nan	7	nan
10	6	nan	nan	10	nan	8	8	8	8	70	8	8	8	nan	8	nan
8	7	nan	nan	23	nan	7	7	19	7	137	7	7	7	nan	7	nan
9	7	nan	nan	8	nan	8	8	14	8	290	8	8	8	nan	14	nan
10	7	nan	nan	9	nan	9	9	11	9	67	11	11	9	nan	9	nan

Table E.10 – Number actions it took for all agents with the symptom solution to reach the goal from every maze position. All “nan" values are positions from which the goal was not reached within 525 actions.

Position		Combined; Participants														
x	y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	nan	22	22	22	22	24	24	nan	nan	22	22	22	24	22	22
1	0	nan	21	21	21	21	23	23	nan	nan	21	21	21	23	21	21
2	0	nan	20	20	20	20	22	22	nan	nan	20	20	20	22	20	20
3	0	nan	19	19	19	19	21	21	nan	nan	19	19	19	21	19	19
4	0	nan	18	18	18	18	20	20	nan	nan	18	18	18	20	18	18
5	0	nan	17	17	17	17	19	19	nan	nan	17	17	17	19	17	17
6	0	nan	16	16	16	16	18	18	nan	nan	16	16	16	18	16	16
7	0	nan	15	15	15	15	17	17	nan	nan	15	15	15	17	15	15
8	0	nan	14	14	14	14	16	16	nan	nan	14	14	14	16	14	14
9	0	nan	13	13	13	13	15	15	nan	13	13	13	13	15	13	13
10	0	nan	14	nan	nan	14	16	16	nan	14	14	14	14	16	nan	nan
9	1	nan	12	12	12	12	14	14	nan	12	12	12	12	14	12	12
10	1	nan	13	nan	nan	13	15	15	nan	13	13	13	13	15	nan	nan
2	2	6	6	nan	6	6	6	6	nan	6	6	6	6	6	6	6
3	2	5	5	5	5	5	5	5	nan	5	5	5	5	5	5	5
4	2	4	4	4	4	4	4	4	nan	4	4	4	4	4	4	4
5	2	3	3	3	3	3	3	3	nan	3	3	3	3	3	3	3
6	2	2	2	2	2	2	2	2	nan	2	2	2	2	2	2	2
7	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	2	nan	11	11	11	11	13	13	nan	11	11	11	11	13	11	11
2	3	nan	7	nan	7	7	7	9	nan	7	7	7	7	9	7	7
6	3	1	1	1	1	1	3	3	nan	1	1	1	1	3	1	1
9	3	nan	10	10	10	10	12	12	nan	10	10	10	10	12	10	10
2	4	nan	6	nan	6	6	8	8	nan	6	6	6	6	8	6	6
3	4	nan	5	5	5	5	7	7	nan	5	5	5	5	7	5	5
4	4	nan	4	4	4	4	6	6	nan	4	4	4	4	6	4	4
5	4	nan	3	3	3	3	5	5	nan	3	3	3	3	5	3	3
6	4	nan	2	2	2	2	4	4	nan	2	2	2	2	4	2	2
7	4	nan	1	1	1	3	1	1	1	1	1	1	1	1	1	1
9	4	nan	9	9	9	9	11	11	nan	9	9	9	9	11	9	9
4	5	nan	5	5	5	5	7	7	nan	5	5	5	5	7	5	5
6	5	nan	3	3	3	3	5	5	nan	3	3	3	3	5	3	3
9	5	nan	8	8	8	8	10	10	nan	8	8	8	8	10	8	8
2	6	nan	8	8	8	8	10	10	nan	8	8	8	8	10	8	8
3	6	nan	7	7	7	7	9	9	nan	7	7	7	7	9	7	7
4	6	nan	6	6	6	6	8	8	nan	6	6	6	6	8	6	6
5	6	nan	5	5	5	5	7	7	nan	7	5	5	5	7	5	5
6	6	nan	4	4	4	4	6	6	nan	4	4	4	4	6	4	4
7	6	nan	5	5	5	5	7	7	nan	5	5	5	5	7	5	5
8	6	nan	6	6	6	6	8	8	nan	6	6	6	6	8	6	6
9	6	nan	7	7	7	7	9	9	nan	7	7	7	7	9	7	7
10	6	nan	8	8	8	8	10	10	nan	8	8	8	8	10	8	8
8	7	nan	7	7	7	nan	9	9	nan	7	7	7	7	9	7	7
9	7	nan	8	8	8	nan	10	10	nan	8	8	8	8	10	8	8
10	7	nan	9	9	9	nan	11	11	nan	9	9	9	9	11	9	9

Table E.11 – Number actions it took for all agents with both solutions to reach the goal from every maze position. All “nan" values are positions from which the goal was not reached within 525 actions.

E.4.2 Obstructed maze

Position		Baseline; Participants														
x	y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	28	nan	nan	nan	nan	nan	nan	28	nan	nan	nan	nan	nan	nan	nan
1	0	25	nan	nan	nan	nan	nan	nan	29	nan	nan	nan	nan	nan	nan	nan
2	0	24	nan	nan	nan	nan	nan	nan	24	nan	nan	nan	nan	nan	nan	nan
3	0	27	nan	nan	nan	nan	nan	nan	31	nan	nan	nan	nan	nan	nan	nan
4	0	22	nan	nan	nan	nan	nan	nan	26	nan	nan	nan	nan	nan	nan	nan
5	0	25	nan	nan	nan	nan	nan	nan	21	nan	nan	nan	nan	nan	nan	nan
6	0	24	nan	nan	nan	nan	nan	nan	34	nan	nan	nan	nan	nan	nan	nan
7	0	23	nan	nan	nan	nan	nan	nan	21	nan	nan	nan	nan	nan	nan	nan
8	0	18	nan	nan	nan	nan	nan	nan	18	nan	nan	nan	nan	170	nan	nan
9	0	23	nan	nan	nan	nan	nan	nan	19	119	nan	nan	nan	nan	nan	nan
10	0	20	nan	nan	nan	nan	nan	nan	18	nan	nan	nan	nan	nan	nan	nan
9	1	20	nan	nan	nan	nan	nan	nan	18	nan	nan	nan	nan	nan	nan	nan
10	1	21	nan	nan	nan	nan	nan	nan	17	nan	nan	131	nan	nan	nan	nan
2	2	14	12	84	nan	34	12	nan	8	56	36	38	330	220	28	16
3	2	11	17	27	nan	39	43	nan	29	11	31	15	5	85	11	15
4	2	6	8	20	nan	16	8	nan	8	28	56	16	6	12	16	10
5	2	3	11	23	nan	5	5	11	7	3	29	41	7	13	3	11
6	2	6	8	12	nan	6	2	nan	4	2	4	2	4	8	2	8
7	2	1	1	1	1	7	29	3	1	5	nan	5	1	1	3	5
9	2	15	nan	nan	nan	nan	nan	nan	15	nan	nan	nan	nan	nan	nan	nan
2	3	11	27	171	nan	51	nan	nan	11	59	15	49	17	7	25	13
6	3	1	9	7	nan	5	125	1	1	1	3	5	33	1	1	3
9	3	16	nan	nan	nan	nan	nan	nan	20	nan	nan	nan	nan	nan	nan	nan
2	4	6	6	54	nan	138	22	nan	14	186	nan	36	20	42	8	8
3	4	5	9	95	nan	99	111	nan	5	59	nan	29	415	55	5	5
4	4	4	16	78	nan	22	24	nan	4	8	18	68	316	14	12	10
5	4	3	21	287	nan	7	87	3	3	nan	3	9	5	11	5	11
6	4	2	6	6	nan	2	4	2	2	40	6	nan	8	98	2	2
7	4	3	1	3	nan	3	1	1	1	1	nan	1	9	11	5	1
9	4	23	nan	nan	nan	nan	nan	nan	13	nan	nan	221	nan	nan	nan	nan
4	5	9	9	25	nan	nan	69	nan	5	nan	33	59	39	21	15	27
9	5	14	nan	nan	nan	42	320	nan	14	142	nan	206	198	250	nan	20
2	6	12	24	64	nan	nan	62	nan	10	90	10	112	234	nan	nan	31
3	6	7	77	177	nan	nan	395	nan	7	15	nan	51	nan	265	nan	12
4	6	6	58	70	nan	98	40	nan	6	nan	nan	20	202	nan	nan	nan
5	6	11	17	147	nan	nan	19	nan	11	29	nan	195	17	205	nan	nan
6	6	8	16	nan	nan	22	234	nan	10	nan	nan	82	216	nan	nan	nan
7	6	13	41	nan	nan	nan	473	nan	15	nan	nan	165	59	257	nan	nan
8	6	12	38	nan	nan	106	120	nan	10	224	nan	128	114	116	nan	nan
9	6	11	15	nan	nan	nan	417	nan	11	nan	nan	nan	355	nan	nan	nan
10	6	14	nan	nan	nan	nan	86	nan	14	nan	nan	nan	490	nan	nan	nan
8	7	11	57	nan	nan	107	181	nan	15	nan	nan	363	317	193	nan	nan
9	7	22	12	nan	nan	nan	nan	nan	14	184	nan	212	168	46	nan	nan
10	7	13	nan	nan	nan	109	145	nan	13	nan	nan	269	227	251	nan	nan

Table E.12 – Number actions it took for all baseline agents to reach the goal from every position in the obstructed maze. All “nan" values are positions from which the goal was not reached within 525 actions.

Position		Preventive; Participants														
x	y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	26	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	26
1	0	25	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	25
2	0	24	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	24
3	0	23	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	23
4	0	22	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	22
5	0	21	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	21
6	0	20	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	20
7	0	19	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	19
8	0	18	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	18
9	0	17	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	17
10	0	18	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
9	1	16	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	16
10	1	17	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
2	2	6	6	6	6	6	6	nan	6	6	nan	nan	6	6	6	6
3	2	5	5	5	5	5	5	nan	5	5	nan	nan	5	5	5	5
4	2	4	4	4	4	4	4	nan	4	4	nan	4	4	4	4	4
5	2	3	3	3	3	3	3	nan	3	3	nan	3	3	3	3	3
6	2	2	2	2	2	2	2	nan	2	2	nan	2	2	2	2	2
7	2	1	1	1	1	1	1	1	1	1	nan	1	1	1	1	1
9	2	15	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	15
2	3	7	9	nan	nan	nan	nan	nan	nan	7	nan	nan	9	nan	9	7
6	3	1	3	1	1	1	1	nan	nan	nan	nan	1	3	nan	3	1
9	3	14	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	14
2	4	6	8	nan	nan	nan	nan	nan	nan	8	nan	nan	8	nan	8	6
3	4	5	7	nan	nan	nan	nan	nan	nan	nan	nan	nan	7	nan	7	5
4	4	4	6	nan	nan	nan	nan	nan	nan	nan	nan	nan	6	nan	6	4
5	4	3	5	nan	nan	3	nan	nan	nan	nan	nan	nan	5	nan	5	3
6	4	2	4	nan	nan	2	nan	nan	nan	nan	nan	nan	4	nan	4	2
7	4	1	1	nan	nan	1	nan	1	1	1	nan		1	1	1	1
9	4	13	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	13
4	5	5	7	nan	nan	nan	nan	nan	nan	nan	nan	nan	7	nan	7	5
9	5	12	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	12
2	6	8	10	nan	nan	nan	nan	nan	nan	nan	nan	nan	10	nan	10	8
3	6	7	9	nan	nan	nan	nan	nan	nan	nan	nan	nan	9	nan	9	7
4	6	6	8	nan	nan	nan	nan	nan	nan	nan	nan	nan	8	nan	8	6
5	6	7	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	7
6	6	8	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	8
7	6	9	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	9
8	6	10	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	10
9	6	11	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	11
10	6	12	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	12
8	7	11	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	11
9	7	12	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	12
10	7	13	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	13

Table E.13 – Number actions it took for all agents with the preventive solution to reach the goal from every position in the obstructed maze. All “nan" values are positions from which the goal was not reached within 525 actions.

Position		Symptom; Participants														
x	y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	36	nan	nan	nan	nan	nan
1	0	nan	nan	nan	nan	nan	nan	nan	nan	347	33	nan	nan	nan	nan	nan
2	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	28	nan	nan	nan	nan	nan
3	0	nan	nan	nan	nan	nan	nan	nan	nan	393	35	nan	nan	nan	nan	nan
4	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	32	nan	nan	nan	nan	nan
5	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	41	nan	nan	nan	nan	nan
6	0	nan	nan	nan	nan	nan	nan	nan	nan	326	58	nan	nan	nan	nan	nan
7	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	63	nan	nan	nan	nan	nan
8	0	nan	nan	nan	nan	nan	nan	nan	nan	452	20	nan	nan	nan	nan	nan
9	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	17	nan	nan	nan	nan	nan
10	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	40	nan	nan	nan	nan	nan
9	1	nan	nan	nan	nan	nan	nan	nan	nan	362	38	nan	nan	nan	nan	nan
10	1	nan	nan	nan	nan	nan	nan	nan	nan	227	23	nan	nan	nan	nan	nan
2	2	6	8	6	nan	42	16	8	8	14	20	12	6	nan	10	nan
3	2	5	39	5	nan	5	47	41	11	5	7	7	5	nan	85	nan
4	2	4	18	4	nan	4	4	10	26	6	8	10	4	nan	18	nan
5	2	3	11	3	nan	3	3	9	3	29	21	3	3	nan	11	nan
6	2	2	2	2	nan	2	2	2	2	84	2	2	2	nan	20	nan
7	2	1	1	1	1	1	1	3	1	1	1	1	1	nan	7	nan
9	2	nan	nan	nan	nan	nan	nan	nan	nan	nan	57	nan	nan	nan	nan	nan
2	3	9	7	7	nan	59	33	13	39	27	33	17	7	nan	23	nan
6	3	3	3	1	nan	1	1	1	1	1	1	1	1	nan	1	1
9	3	nan	nan	nan	nan	nan	nan	nan	nan	52	52	nan	nan	nan	nan	nan
2	4	8	10	6	nan	16	18	6	6	266	6	10	6	nan	8	nan
3	4	5	7	5	nan	7	5	5	19	13	9	5	5	nan	5	nan
4	4	4	6	4	nan	4	8	4	4	46	4	4	4	nan	10	nan
5	4	3	11	3	nan	3	5	3	3	237	3	3	3	nan	3	nan
6	4	2	4	2	nan	2	2	2	2	4	2	2	2	nan	2	nan
7	4	1	1	1	1	3	1	1	1	7	3	1	1	nan	1	1
9	4	nan	nan	nan	nan	nan	nan	nan	nan	285	27	nan	nan	nan	nan	nan
4	5	7	7	5	nan	nan	nan	5	5	7	9	nan	5	nan	7	nan
9	5	nan	nan	nan	nan	nan	nan	nan	nan	nan	20	nan	nan	nan	nan	nan
2	6	12	18	22	nan	nan	nan	8	8	158	38	nan	nan	nan	nan	nan
3	6	9	11	27	nan	nan	nan	7	9	29	13	nan	nan	nan	nan	nan
4	6	8	16	8	nan	nan	nan	6	6	22	34	nan	nan	nan	nan	nan
5	6	nan	nan	nan	nan	nan	nan	nan	nan	47	9	nan	nan	nan	nan	nan
6	6	nan	nan	nan	nan	nan	nan	nan	nan	nan	26	nan	nan	nan	nan	nan
7	6	nan	nan	nan	nan	nan	nan	nan	nan	nan	17	nan	nan	nan	nan	nan
8	6	nan	nan	nan	nan	nan	nan	nan	nan	298	14	nan	nan	nan	nan	nan
9	6	nan	nan	nan	nan	nan	nan	nan	nan	nan	51	nan	nan	nan	nan	nan
10	6	nan	nan	nan	nan	nan	nan	nan	nan	nan	18	nan	nan	nan	nan	nan
8	7	nan	nan	nan	nan	nan	nan	nan	nan	101	15	nan	nan	nan	nan	nan
9	7	nan	nan	nan	nan	nan	nan	nan	nan	76	14	nan	nan	nan	nan	nan
10	7	nan	nan	nan	nan	nan	nan	nan	nan	nan	37	nan	nan	nan	nan	nan

Table E.14 – Number actions it took for all agents with the symptom solution to reach the goal from every position in the obstructed maze. All “nan" values are positions from which the goal was not reached within 525 actions.

Position		Combined; Participants														
x	y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	26	nan	nan	nan
1	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	25	nan	nan	nan
2	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	24	nan	nan	nan
3	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	23	nan	nan	nan
4	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	22	nan	nan	nan
5	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	21	nan	nan	nan
6	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	20	nan	nan	nan
7	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	19	nan	nan	nan
8	0	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	18	nan	nan	nan
9	0	nan	nan	nan	nan	nan	nan	nan	nan	17	nan	nan	17	nan	nan	nan
10	0	nan	nan	nan	nan	nan	nan	nan	nan	18	nan	nan	18	nan	nan	nan
9	1	nan	nan	nan	nan	nan	nan	nan	nan	16	nan	nan	16	nan	nan	nan
10	1	nan	nan	nan	nan	nan	nan	nan	nan	17	nan	nan	17	nan	nan	nan
2	2	6	6	nan	6	6	6	6	nan	6	6	6	6	6	6	6
3	2	5	5	5	5	5	5	5	nan	5	5	5	5	5	5	5
4	2	4	4	4	4	4	4	4	nan	4	4	4	4	4	4	4
5	2	3	3	3	3	3	3	3	nan	3	3	3	3	3	3	3
6	2	2	2	2	2	2	2	2	nan	2	2	2	2	2	2	2
7	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	2	nan	nan	nan	nan	nan	nan	nan	nan	15	nan	nan	15	nan	nan	nan
2	3	nan	7	nan	7	7	7	9	nan	7	7	7	7	9	7	7
6	3	1	1	1	1	1	3	3	nan	1	1	1	1	3	1	1
9	3	nan	nan	nan	nan	nan	nan	nan	nan	14	nan	nan	14	nan	nan	nan
2	4	nan	6	nan	6	6	8	8	nan	6	6	6	6	8	6	6
3	4	nan	5	5	5	5	7	7	nan	5	5	5	5	7	5	5
4	4	nan	4	4	4	4	6	6	nan	4	4	4	4	6	4	4
5	4	nan	3	3	3	3	5	5	nan	3	3	3	3	5	3	3
6	4	nan	2	2	2	2	4	4	nan	2	2	2	2	4	2	2
7	4	nan	1	1	1	3	1	1	1	1	1	1	1	1	1	1
9	4	nan	nan	nan	nan	nan	nan	nan	nan	13	nan	nan	13	nan	nan	nan
4	5	nan	5	5	5	5	7	7	nan	5	5	5	5	7	5	5
9	5	nan	nan	nan	nan	nan	nan	nan	nan	12	nan	nan	12	nan	nan	nan
2	6	nan	8	8	nan	8	10	10	nan	8	nan	8	8	nan	8	8
3	6	nan	7	7	nan	7	9	9	nan	7	nan	7	7	nan	7	7
4	6	nan	6	6	nan	6	8	8	nan	6	nan	6	6	nan	6	6
5	6	nan	nan	nan	nan	nan	nan	nan	nan	7	nan	nan	7	nan	nan	nan
6	6	nan	nan	nan	nan	nan	nan	nan	nan	8	nan	nan	8	nan	nan	nan
7	6	nan	nan	nan	nan	nan	nan	nan	nan	9	nan	nan	9	nan	nan	nan
8	6	nan	nan	nan	nan	nan	nan	nan	nan	10	nan	nan	10	nan	nan	nan
9	6	nan	nan	nan	nan	nan	nan	nan	nan	11	nan	nan	11	nan	nan	nan
10	6	nan	nan	nan	nan	nan	nan	nan	nan	12	nan	nan	12	nan	nan	nan
8	7	nan	nan	nan	nan	nan	nan	nan	nan	11	nan	nan	11	nan	nan	nan
9	7	nan	nan	nan	nan	nan	nan	nan	nan	12	nan	nan	12	nan	nan	nan
10	7	nan	nan	nan	nan	nan	nan	nan	nan	13	nan	nan	13	nan	nan	nan

Table E.15 – Number actions it took for all agents with both the solutions to reach the goal from every position in the obstructed maze. All “nan" values are positions from which the goal was not reached within 525 actions.

Appendix F

Additional Results

F.1 Number actions taken during training

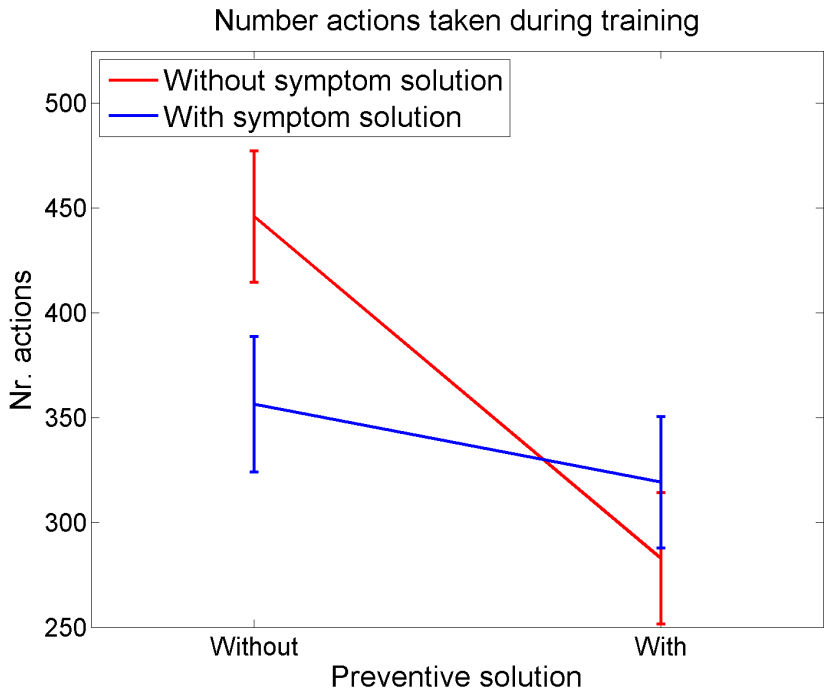
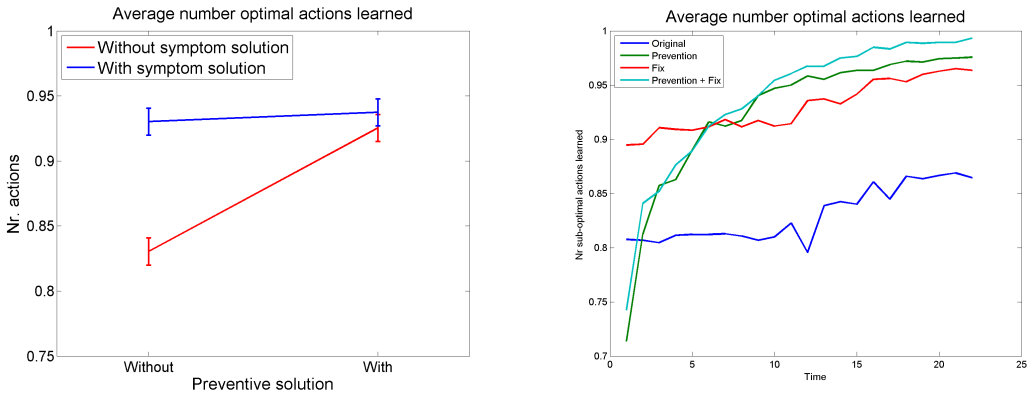


Figure F.1 – The interaction plot of the number of actions taken during training. The error bars are plotted using the standard errors.

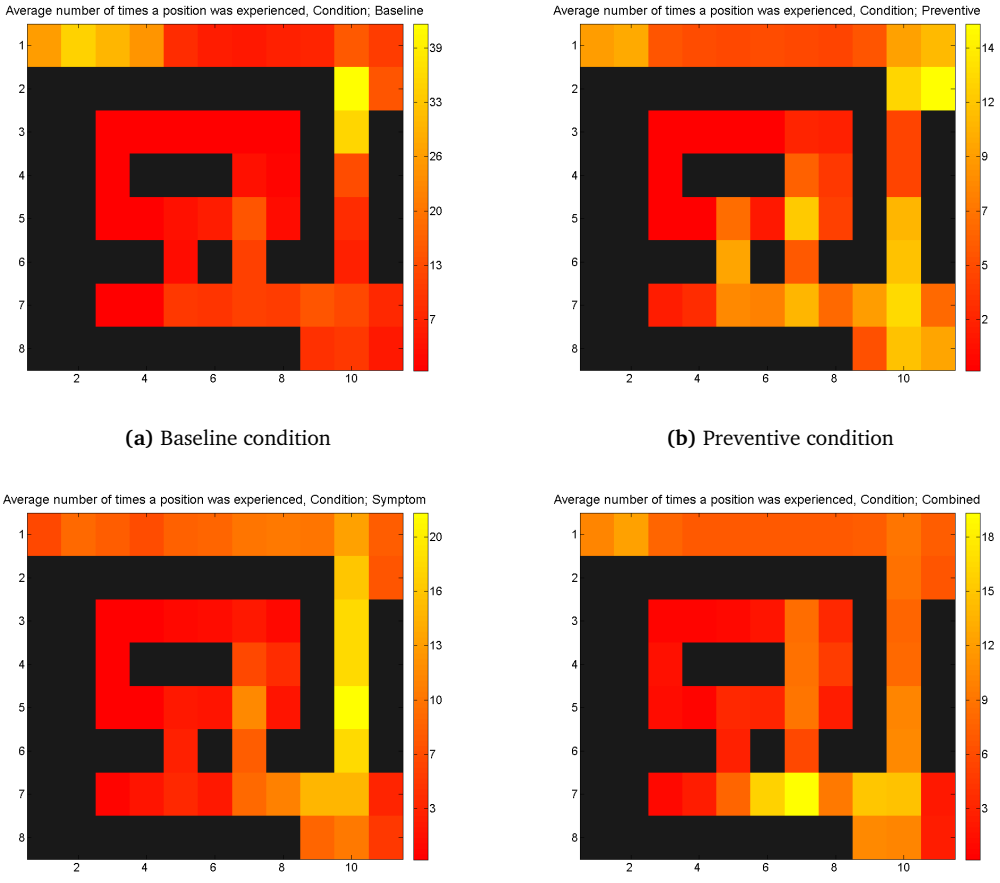
F.2 Number of optimal actions learned



(a) **(b)**

Figure F.2 – These maze overviews show the number of optimal actions learned by each condition on average. a) This interaction plot shows the average optimal actions learned over the entire training time (error bars based on standard errors). b) This graph shows the number of optimal actions learned over time based on simulated agents stored every 25 actions for a total of 22 times.

F.3 Number of times each maze position was experienced



(a) Baseline condition

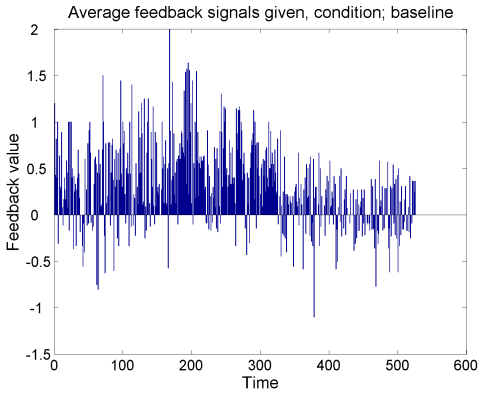
(b) Preventive condition

(c) Symptom condition

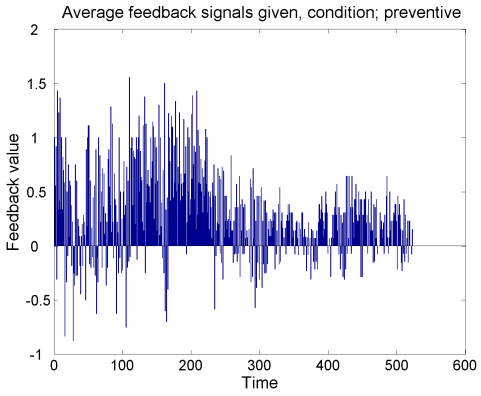
(d) Combined condition.

Figure F.3 – The average number of times each maze position was experienced during training for each of the four conditions.

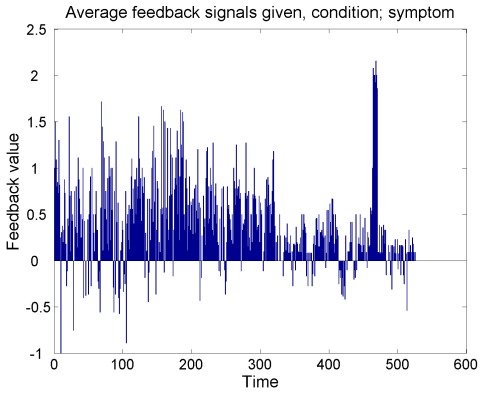
F.4 Feedback signals given on average



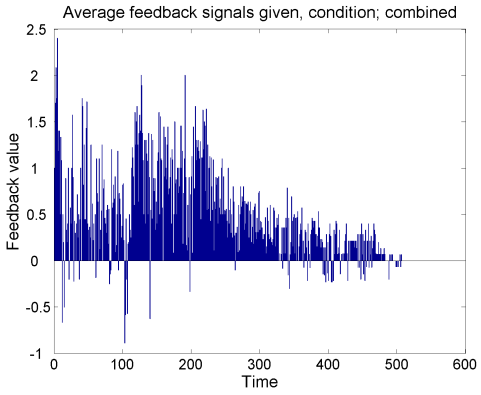
(a) Baseline condition.



(b) Preventive condition.



(c) Symptom condition.



(d) Combined condition.

Figure F.4 – These four bar plots show the average feedback signals given for each condition. The peak in the combined condition is caused by one participant providing the agent with more than 15 positive feedback signals per performed action for a short period of time. These plots give a first indication of how users adjust their feedback strategy over time per condition. Note that each bar may consist of a different number of participants; as time progresses, more participants finished training. This is especially the case for the preventive, symptom and combined conditions.

F.5 Level of frustration to teach the agent

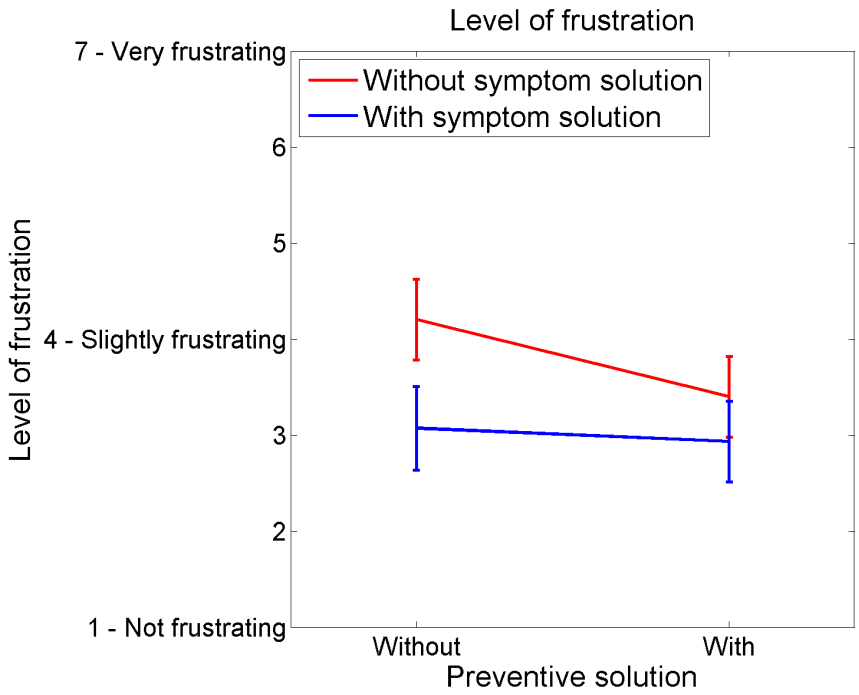


Figure F.5 – This interaction plot shows the average level of frustration as reported by the participants while teaching the agent. The error bars are based on the standard errors.

F.6 Response to feedback

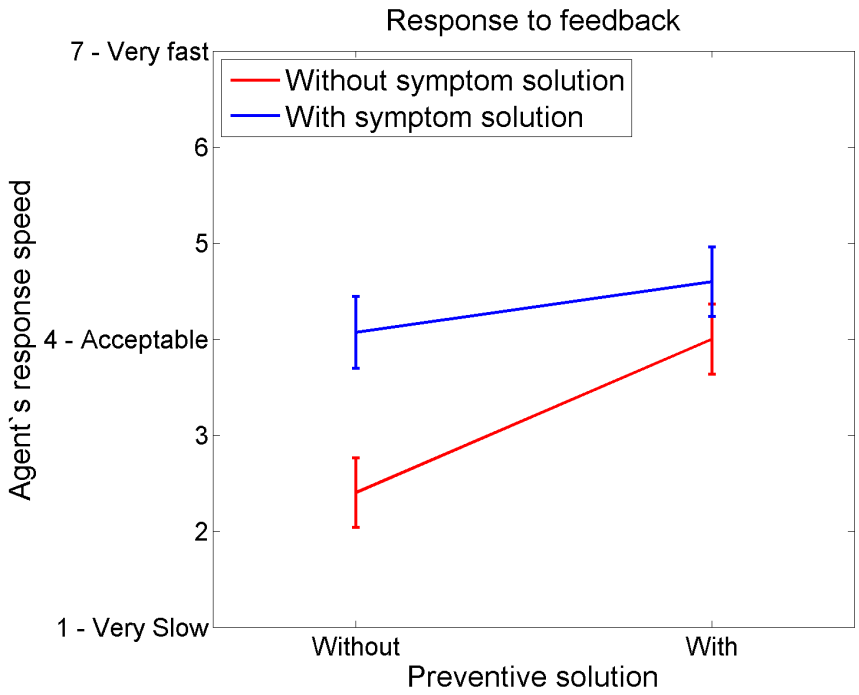


Figure F.6 – The interaction plot shows how quick the agent responded to the feedback according to the participants.

F.7 Remarks from participants

Participant	Remark
2	“Quite difficult."
3	“Goodluck with this."
6	“Peter got stuck between two squares. It seems it exploited too much and explored too little."
10	“Goodluck with the study, but Peter won’t learn this."
13	“I got stuck in the top bar."
15	“Goodluck with it."

(a) Baseline condition.

Participant	Remark
2	“This was fun! :D"
3	“The first time went quite slow. After that he was quite quickly at the goal."
5	“At times he [Peter] seemed to get stuck on a spot where he would switch between two squares."
9	“The first time to reach the goal took some time but he [Peter] learns really good after the first time already! :D"
14	“Peter is quite a dumn thing... It kept switching between two squares. At some point I just stopped giving feedback or started to give only negative feedback."

(b) Preventive condition.

Participant	Remark
2	“It didn’t work! :(Crazy Peter refused to listen!"
4	“He [Peter] seemed to do it better at the start than later on. The constant switchig between squares became more frequent when he tried to find a path more often."
10	“After four complete trials, he [Peter] could almost do it perfectly. From experience I know that is fast, and I’m impressed."

(c) Symptom condition.

Participant	Remark
3	“Super fun! :)"
9	“A very fun experiment!"
14	“Fun!"

(d) Combined condition.

Table F.1 – Participants had the option to give remarks. This table shows the remarks that applied to the agent and task (translated from Dutch).

Bibliography

A. G. Barto, Reinforcement learning: An introduction, MIT press, 1998.

W. B. Knox, Learning from human-generated reward .

C. J. C. H. Watkins, Learning from delayed rewards, Ph.D. thesis, University of Cambridge England, 1989.

L. Baird, Residual algorithms: Reinforcement learning with function approximation, in: Proceedings of the twelfth international conference on machine learning, 30–37, 1995.

R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, E. Wiewiora, Fast gradient-descent methods for temporal-difference learning with linear function approximation, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 993–1000, 2009a.

R. S. Sutton, H. R. Maei, C. Szepesvári, A Convergent $O(n)$ Temporal-difference Algorithm for Off-policy Learning with Linear Function Approximation, in: Advances in neural information processing systems, 1609–1616, 2009b.