References

[1] Luiz Pessoa,“Cognition and Emotion”: Scholarpedia, 2009. [Online]. Available: http://www.scholarpedia.org/article/Cognition_and_emotion [Accessed: December 2013]

[2] Ethem Alpaydin, Lecture slides for Introduction to machine learning, The MIT Press 2004. [Online], Available: http://www.cmpe.boun.edu.tr/~ethem/i2ml [Accessed: December 2013]

[3] Stuart Russell and Peter Norvig, Artificial intelligence a modern approach, Third edition, Pearson Education, Inc , 2010, Chapter 21

[4] Jacques Fleuriot, Intelligent Agents and their Environments, School of informatics, University of Edinburgh, [E-book] Available: http://www.inf.ed.ac.uk/teaching/courses/inf2d/timetable/01_Intelligent_Agents.pdf

[5] S.R.K.Branavan, David Silver and Reginna Barzilay. Learning to win by Reading manuals in a Monte-Carlo Framework in Journal of Artificial intelligence Research43 (2012) 661-704, AI Access Foundation, 2012.

[6] S.R.K.Branavan, Harr Chen, Luke S.Zettlemoyer and Reginna Barzilay. Reinforcment learning for mapping instructions to actions. CSAIL – MIT

[7] S.R.K.Branavan, Nate Kushman, Tao Lei and Reginna Barzilay. Learning High-level Planning from Text. CSAIL – MIT

[8] James Timothy Oates. Grounding knowledge in sensors: Unsupervised learning for languages and planning. PhD thesis, University of Massachusetts Amherst, 2001

[9] Deb K.Roy and Alex P. Pentland. Leaning words from sights and sounds: a computational model. Cognitive Science 26, Pages 113-146 , 2002

[10] Chen Yu and Dana H. Ballard. On the integration of grounding language and learning objects. Department of Computer Science - University of Rochester

[11] Michael Fleischman and Deb Roy. Intentional context in situated natural language learning. In Proceedings of CoNLL, pages 104-111, 2005.

[12] David L. Chen and Raymond J. Mooney. Learning to sportscast: a test of grounded language acquisition. In Proceedings of ICML, 2008.

[13] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Understaning natural language commands for robotic navigation and mobile manipulation. In Proceedings of 25th AAAI Conference on Artificial Intelligence.

[14] Mortaze Zolfpour Arokhlo, Ali Selamat, Siti Zaiton Mohd Hashim, Md Hafiz Selamat. Multi-agent Reinforcement Learning for Route Guidance System International Journal of Advancements in Computing Technology Volume 3, Number 6, July 2011

[15] Jeffrey Lou Adler, Victor J. Blue, "A Cooperative Multi-agent Transportation Management and Route Guidance System", Transportation Research Part C, Vol.10, No.5, pp.433-454, 2002.

[16] Francisco Martinez-Gil, Miguel Lozano and Fernando Fernandez, "Emergent collective behaviors in a multi-agent reinforcement learning pedestrian simulation: a case study", Departament d'Inform`atica. Universitat de Val`encia. Av. de la Universidad s/n, 46100 Burjassot, Valencia, Spain

[17] Francisco Martinez-Gil, Miguel Lozano and Fernando Fernandez, "Multi-agent reinforcement learning for simulating pedestrian navigation", Departament d'Inform`atica. Universitat de Val`encia. Av. de la Universidad s/n,46100 Burjassot, Valencia, Spain

[18] Michelle McPartland and Marcus Gallagher, “Learning to be a Bot: Reinforcment Learning in Shooter Game”, Artificial Intelligence and Interactive Digital Entertainment Conference, October 22–24, Stanford, California, 2008.

[19] Sanchez-Crespo, Dalmau.D, Core Techniques and Algorithms in Game Programming. Indianapolis, Indiana: New Riders, 2003.

[20] Bradley.J, and Hayes. G, “Group Utility Functions: Learning Equilibria between Groups of Agents in Computer Games By Modifying the Reinforcement Signal”, Congress on Evolutionary Computation, 2005.

[21] Manslow. J, “Using Reinforcement Learning to Solve AI Control Problems, in AI Game Programming Wisdom 2”, S. Rabin, (Editor). Hingham, USA: Charles River Media, 2004.

[22] Merrick.K and Maher.M.L, “Motivated Reinforcement Learning for Non-Player Characters in Persistent Computer Game Worlds”. In ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, Los Angeles, USA, 2006.

[23] Luca M. Gambardella and Marco Dorigo, “Ant-Q: A Reinforcement Learning approach to the travelling salesman problem”,In Proceedings of ML-95, Twelfth Intern. Conf. on Machine Learning, Morgan Kaufmann, 1995, 252–260.

[24] Dorigo M., V.Maniezzo and A.Colorni, “The Ant System: Optimization by a colony of cooperating agents”, IEEE Transactions on Systems: Man and Cybernetics, 26, 2, in press, 1996.

[25]Sascha Lange, Martin Riedmiller and Arne Voigtlander, "Autonomous reinforcement learning on raw visual input data in a real world application", In International Joint Conference on Neural Networks. Brisbane, Australia, 2012

[26] D. Ernst, P. Geurts, and L. Wehenkel, “Tree-Based Batch Mode Reinforcement Learning,” Journal of Machine Learning Research, vol. 6,no. 1, pp. 503–556, 2006.

[27] R. Hafner and M. Riedmiller, “Reinforcement learning in feedback control,” Machine Learning, vol. 27, no. 1, pp. 55–74, 2011, 10.1007/s10994-011-5235-x. [Online]. Available: http://dx.doi.org/10.1007/s10994-011-5235-x

[28] R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun, “Deep belief net learning in a long-range vision system for autonomous off-road driving,” in IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008, 2008, pp. 628–633.

[29] G. Gordon, “Stable Function Approximation in Dynamic Programming,” in Proceedings of the 12th Int. Conf. on Machine Learning (ICML), 1995, pp.261–268.

[30] D. Ernst, R. Maree, and L. Wehenkel, “Reinforcement learning with raw pixels as input states,” in Workshop on Intelligent Computing in Pattern Analysis/Synthesis (IWICPAS), 2006, pp. 446–454.

[31] S. Jodogne and J. Piater, “Closed-loop learning of visual control policies,” Journal of Artificial Intelligence Research, vol. 28, pp. 349–391, 2007.

[32] S. Jodogne, C. Briquet, and J. Piater, “Approximate Policy Iteration for Closed-Loop Learning of Visual Tasks,” in Proceedings of the European Conference on Machine Learning, 2006.

[33] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction A Bradford Book The MIT PressCambridge, Massachusetts London, England

Appendix A: Intelligence Agent and their behaviour

A.1 Intelligence Agent and their behaviour comparison

This table explains the comparison of intelligence agents and their behaviour. The agents defined here can be falling in to the other environment types as well; here what only considered is the major behaviour of the particular agents.

Task Environment	Observable	Agent	Deterministic	Episodic	Static	Discrete	Benign/Adversarial
Crossword Puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete	Benign
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete	Adversarial
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete	Adversarial
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete	Adversarial
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous	Benign
Medical Diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous	Benign
Image Analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous	Benign
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous	Benign
Interactive English Tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous	Adversarial
Mail-Sorting robot	Fully	Single	Deterministic	Episodic	Dynamic	Continuous	Benign

A.2 Intelligence Agents and the structure of the agents

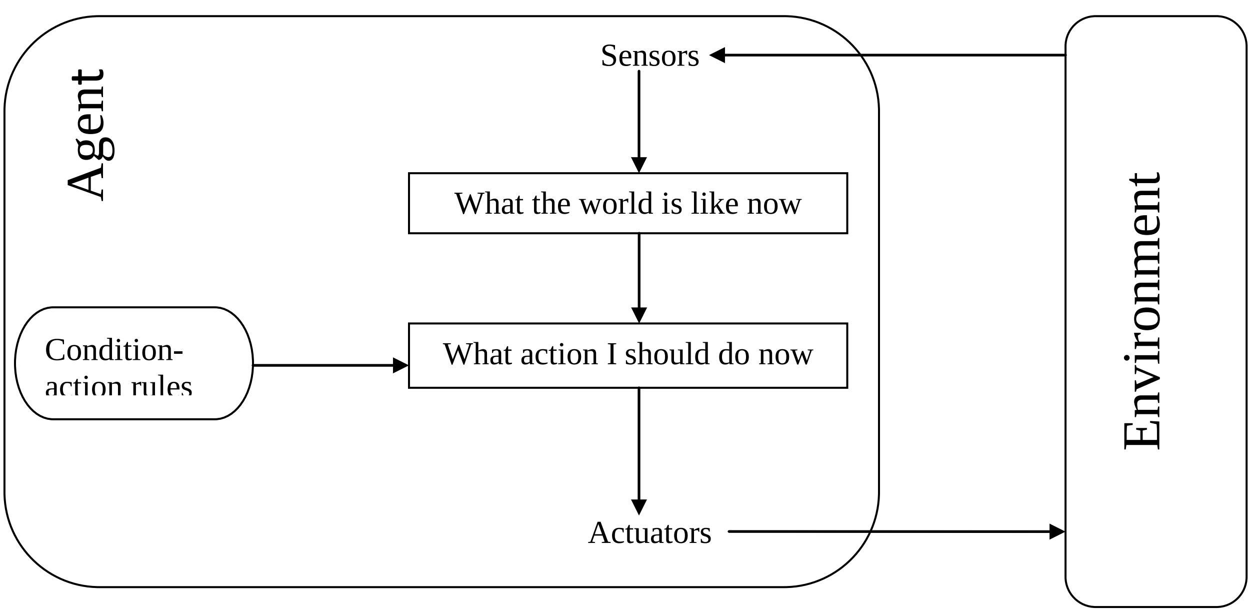
The following details are taken from the book Stuart Russell and Peter Norvig, Artificial intelligence a modern approach [3].

Simple reflex agent

The simplest kind of agent is the simple reflex agent. These agents select actions on the basis of the current percept, ignoring the rest of the percept history Simple reflex behaviours occur even in more complex environments. Imagine yourself as the driver of the automated taxi. If the car in front brakes and its brake lights come on, then you should notice this and initiate braking. In other words, some processing is done on the visual input to establish the condition we call "The car in front is braking." Then, this triggers some established connection in the agent program to the action "initiate braking." We call such a connection a condition-action rule written as;

if car- in-front- is- braking then initiate- braking.

The following diagram shows how the condition-action rules allow the agent to make the connection from percept to action.



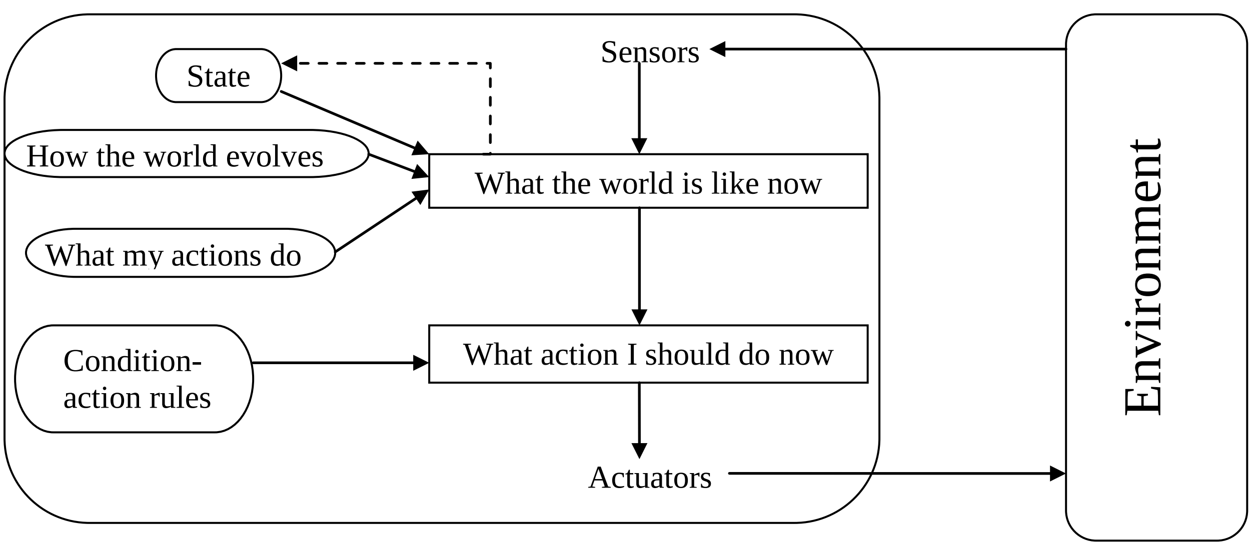
Schematic diagram of Simple Reflex Agent

Humans also have many such connections, some of which are learned responses (as for driving) and some of which are innate reflexes (such as blinking when something approaches the eye). In the course of the book, we show several different ways in which such connections can be learned and implemented.

Model based reflex agent

The most effective way to handle partial observability is for the agent to keep track of the part of the world it can't see now. That is, the agent should maintain some sort of internal state that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state. For the braking problem, the internal state is not too extensive-just the previous frame from the camera, allowing the agent to detect when two red lights at the edge of the vehicle go on or off simultaneously. For other driving tasks such as changing lanes, the agent needs to keep track of where the other cars are if it can't see them all at once.

Regardless of the kind of representation used, it is seldom possible for the agent to determine the current state of a partially observable environment exactly. Instead, the box labelled "what the world is like now" (in the following diagram) represents the agent's "best guess/guesses".

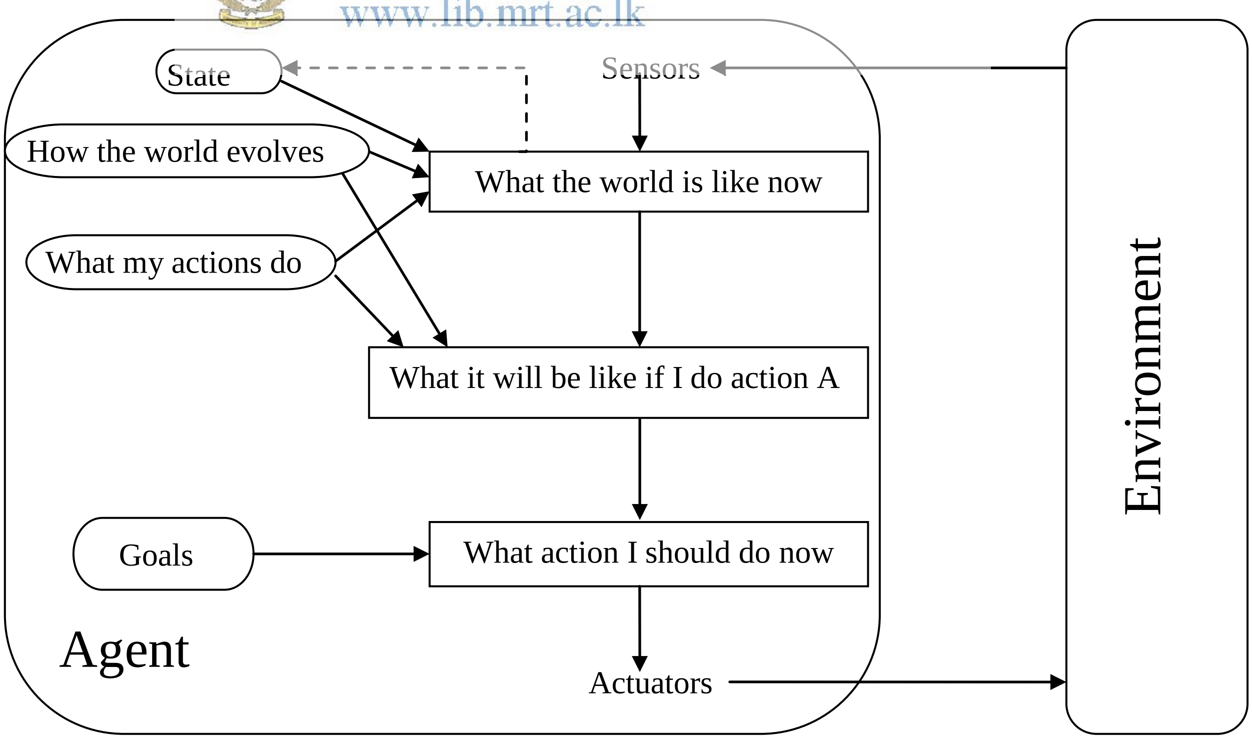


Schematic diagram of Model-based reflex agent

For example, an automated taxi may not be able to see around the large truck that has stopped in front of it and can only guess about what may be causing the hold - up. Thus, uncertainty about the current state may be unavoidable, but the agent still has to make a decision.

Goal-based Agent

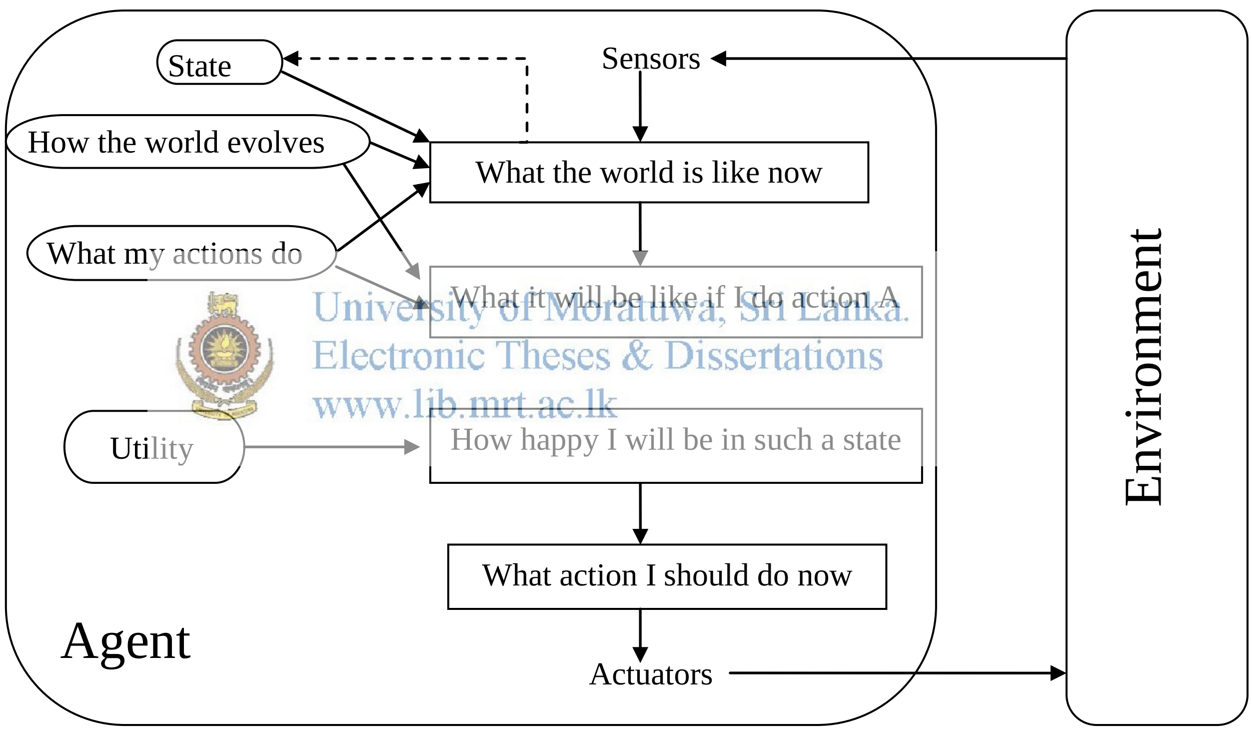
Knowing something about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to. In other words, as well as a current state description, the agent needs some sort of goal information that describes situations that are desirable—for example, being at the passenger's destination. The agent program can combine this with the model (the same information as was used in the model-based reflex agent) to choose actions that achieve the goal. The following diagram shows the goal-based agent's structure.



Goal based agent

Utility-based agents

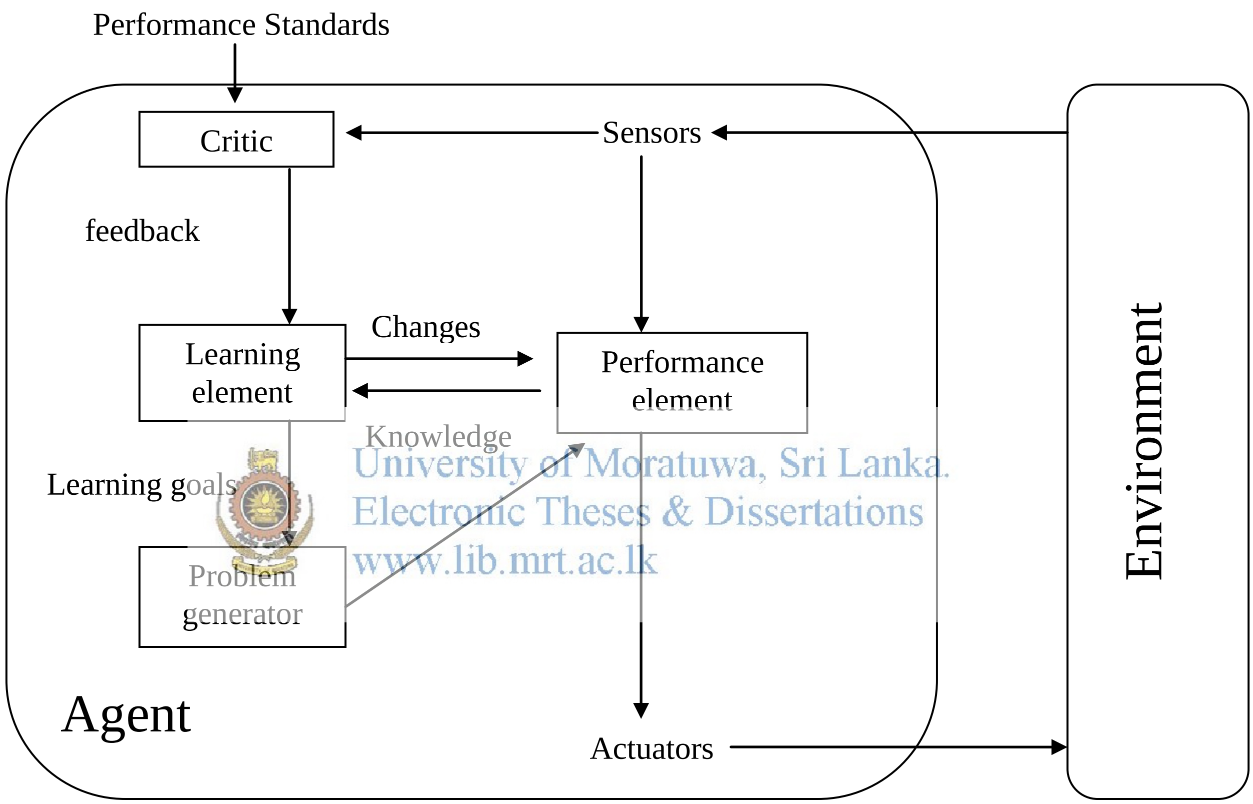
Goals alone are not enough to generate high-quality behaviour in most environments. For example, many action sequences will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others. Goals just provide a crude binary distinction between "happy" and "unhappy" states. A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent. Because "happy" does not sound very scientific, economists and computer scientists use the term utility instead.



Utility-based agent

Learning agent

One of the advantages in learning agent is that it allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow. A learning agent can be divided into four conceptual components as shown as in the following diagram.



A general model of learning agent

The most important distinction is between the learning element, which is responsible for making improvements, and the performance element, which is responsible for selecting external actions. The performance element is what we have previously considered to be the entire agent: it takes in perceptions and decides on actions. The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future.

The design of the learning element depends very much on the design of the performance element. When trying to design an agent that learns a certain capability, the first question is not "How am I going to get it to team this?" but "What kind of performance element will my agent need to do this once it has learned how?" Given an agent design, learning mechanisms can be constructed to improve every part of the agent. The critic tells the learning element how well the agent is doing with respect to a fixed performance standard. The critic is necessary because the percept themselves provide no indication of the agent's success. For example, a chess program could receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so. It is important that the performance standard be fixed. Conceptually, one should think of it as being outside the agent altogether because the agent must not modify it to fit its own behaviour. The last component of the learning agent is the problem generator. It is responsible for suggesting actions that will lead to new and informative experiences.

A.3 Ant-Q algorithm

Ant-Q is an algorithm which is inspired by both Q-learning algorithm and the observation of and colonies behaviour. The following algorithm is an Ant-Q algorithm modelled for Travelling Salesman Problem (TSP) [23] which is explained under the Chapter 2.

```
1./* Initialization phase */
  For each pair (r,s) AQ(r,s):= AQ_0  End-for
   For k:=1 to m do
      Let r_k1 be the starting city for agent k
      J_k(r_k1):= {1, ..., n} - r_k1
      /* J_k(r_k1) is the set of yet to be visited cities for agent k in city r_k1 */
      r_k:=r_k1
      /* r_k is the city where agent k is located */
   End-for
2. /* This is the step in which agents build their tours. The tour of agent k is stored in
  Tour_k. Given that local reinforcement is always null, only the next state evaluation is used
  to update AQ-values. */
  For i:=1 to n do
   If i≠n
     Then
      For k:=1 to m do
         Choose the next city s_k according to formula (1)
         If i≠n-1 Then J_k(s_k):= J_k(r_k) - s_k
         If i=n-1 Then J_k(s_k):= J_k(r_k) - s_k + r_k1
         Tour_k(i):=(r_k,s_k)
      End-for
     Else
      For k:=1 to m do  /* In this cycle all the agents go back to the initial city r_k1 */
         s_k := r_k1
         Tour_k(i):=(r_k,s_k)
      End-for
   For k:=1 to m do
      AQ(r_k,s_k):=(1-α)AQ(r_k,s_k)+α·γ· Max         AQ(s_k,z)
                                         z∈J_k(s_k)
      /* This above is formula (2), where the reinforcement ΔAQ(r_k,s_k) is always null */
      r_k := s_k  /* New city for agent k */
   End-for
  End-for
3. /* In this step delayed reinforcement is computed and AQ-values are updated using formula
  (2), in which the next state evaluation term γ·Max AQ(r_k1,z) is null for all z */
  For k:=1 to m do
   Compute L_k  /*L_k is the length of the tour done by agent k*/
  End-for
  For each edge (r,s)
   Compute the delayed reinforcement ΔAQ(r,s)
  /*The delayed reinforcement ΔAQ(r,s) is a function of L_k's */
  End-for
  Update AQ-values applying a formula (2)
4. If (End_condition = True)
   then Print shortest of L_k
   else goto Step 2
```

A.4 Deep fitted Q algorithm schema

The following algorithm is a general algorithm scheme of Deep fitted Q with the two basic building blocks encoder training and fitting the Q values [25] which is discussed in the subsection 2.3 under chapter 2.

1) **Initialization** Set episode counter $k \leftarrow 0$. Set sample counter $p \leftarrow 0$. Create an initial (random) exploration strategy $\pi^0 : \mathcal{Z} \mapsto A$ and an initial encoder $\phi : \mathcal{S} \mapsto_{W^0} \mathcal{Z}$ with (random) weight vector W^0. Start with an empty set $\mathcal{F}_{\mathcal{S}} = \oslash$ of transitions $(s_t, a_t, r_{t+1}, s_{t+1})$
2) **Episodic Exploration** In each time step t calculate the feature vector z_t from the observed image s_t by using the present encoder $z_t = \phi(s_t; W^k)$. Select an action $a_t \leftarrow \pi^k(z_t)$ and store the completed transition in image space $\mathcal{S}$: $\mathcal{F}_{\mathcal{S}} \leftarrow \mathcal{F}_{\mathcal{S}} \cup (s_p, a_p, r_{p+1}, s_{p+1})$ incrementing p with each observed transition.
3) **Encoder Training** Train an auto-encoder (see [7]) on the p observations in $\mathcal{F}_{\mathcal{S}}$ using RProp during layer-wise pretraining and finetuning. Derive the encoder $\phi(\cdot ; W^{k+1})$ (first half of the auto-encoder). Set $k \leftarrow k + 1$.
4) **Encoding** Apply the encoder $\phi(s; W^k)$ to all transitions $(s_t, a_t, r_{t+1}, s_{t+1}) \in \mathcal{F}_{\mathcal{S}}$, transfering them into the feature space $\mathcal{Z}$, constructing a set $\mathcal{F}_{\mathcal{Z}} = \{(z_t, a_t, r_{t+1}, z_{t+1}) | \; t = 1, \ldots, p\}$ with $z_t = \phi(s_t; W^k)$.
5) **Inner Loop: FQI** Call FQI with $\mathcal{F}_{\mathcal{Z}}$. Starting with an initial approximation $\hat{Q}^0(z, a) = 0 \quad \forall (z, a) \in Z \times A$ FQI (details in [4]) iterates over a dynamic programming (DP) step creating a training set $\mathcal{P}^{i+1} = \{(z_t, a_t; \bar{q}_t^{i+1}) | t = 1, ..., p\}$ with $\bar{q}_t^{i+1} = r_{t+1} + \gamma \max_{a' \in A} \hat{Q}^i(z_{t+1}, a')$ [4] and a supervised learning step training a function approximator on $\mathcal{P}^{i+1}$, obtaining the approximated Q-function $\hat{Q}^{i+1}$. After convergence, the algorithm returns the unique fix-point $\bar{Q}^k$.
6) **Outer loop** If satisfied return approximation $\bar{Q}^k$, greedy policy π and encoder $\phi(\cdot ; W^k)$. Otherwise derive an ϵ-greedy policy π^k from $\bar{Q}^k$ and continue with step 2.

Appendix B:
Major Implementation in Software Development

B.1 Implementation of the RLearner Class

```
import java.util.Vector;
import java.lang.*;
import java.lang.reflect.*;

public class RLearner {

    RLWorld thisWorld;
    RLPolicy policy;

    // Learning types
    public static final int Q_LEARNING = 1;
    public static final int SARSA = 2;
    public static final int Q_LAMBDA = 3; // Good parms were lambda=0.05,
gamma=0.1, alpha=0.01, epsilon=0.1

    // Action selection types
    public static final int E_GREEDY = 1;
    public static final int SOFTMAX = 2;

    int learningMethod;
    int actionSelection;

    double epsilon;
    double temp;

    double alpha;
    double gamma;
    double lambda;

    int[] dimSize;
    int[] state;
    int[] newstate;
    int action;
    double reward;

    int epochs;
        public int epochsdone;

    Thread thisThread;
    public boolean running;

    Vector trace = new Vector();
    int[] saPair;
```

```
long timer;

    boolean random = false;
        Runnable a;

    public RLearner( RLWorld world) {
                // Getting the world from the invoking method.
                thisWorld = world;

                // Get dimensions of the world.
                dimSize = thisWorld.getDimension();

                // Creating new policy with dimensions to suit the world.
                policy = new RLPolicy( dimSize );

                // Initializing the policy with the initial values defined
by the world.
                policy.initValues( thisWorld.getInitValues() );

                learningMethod = Q_LEARNING;  //Q_LAMBDA;//SARSA;
                actionSelection = E_GREEDY;

                // set default values
                epsilon = 0.1;
                temp = 1;

                alpha = 1; // For CliffWorld alpha = 1 is good
                gamma = 0.1;
                lambda = 0.1;  // For CliffWorld gamma = 0.1, l = 0.5
(l*g=0.05)is a good choice.

                System.out.println( "RLearner initialised" );

    }



// execute one trial
        public void runTrial() {
                System.out.println( "Learning! ("+epochs+" epochs)\n" );
                for( int i = 0 ; i < epochs ; i++ ) {
                                if( ! running ) break;

                                runEpoch();

                        if( i % 1000 == 0 ) {
                            // give text output
                            timer = ( System.currentTimeMillis() - timer );
                            System.out.println("Epoch:" + i + " : " +
timer);

                            timer = System.currentTimeMillis();
                        }
                }
        }
```

```
// execute one epoch
        public void runEpoch() {

                // Reset state to start position defined by the world.
                state = thisWorld.resetState();

                switch( learningMethod ) {

                case Q_LEARNING : {

                double this_Q;   double max_Q;    double new_Q;

                        while( ! thisWorld.endState() ) {

                            if( ! running ) break;
                                action = selectAction( state );
                                newstate = thisWorld.getNextState( action );
                                reward = thisWorld.getReward();

                                 this_Q = policy.getQValue( state, action );
                                 max_Q = policy.getMaxQValue( newstate );

                                 // Calculate new Value for Q
                                 new_Q = this_Q + alpha * ( reward + gamma *
max_Q - this_Q );

                                 policy.setQValue( state, action, new_Q );
                                 // Set state to the new state.
                                 state = newstate;
                        }

           }
case SARSA : {

            int newaction; double this_Q; double next_Q;  double new_Q;

            action = selectAction( state );
                while( ! thisWorld.endState() ) {

                    if( ! running ) break;

                    newstate = thisWorld.getNextState( action );
                    reward = thisWorld.getReward();
                    newaction = selectAction( newstate );
                    this_Q = policy.getQValue( state, action );
                    next_Q = policy.getQValue( newstate, newaction );
                    new_Q = this_Q + alpha * ( reward + gamma * next_Q -
this_Q );
                    policy.setQValue( state, action, new_Q );

                    // Set state to the new state and action to the new
action.
                    state = newstate;
                    action = newaction;
                }

        }
```

B.2 Class Diagram