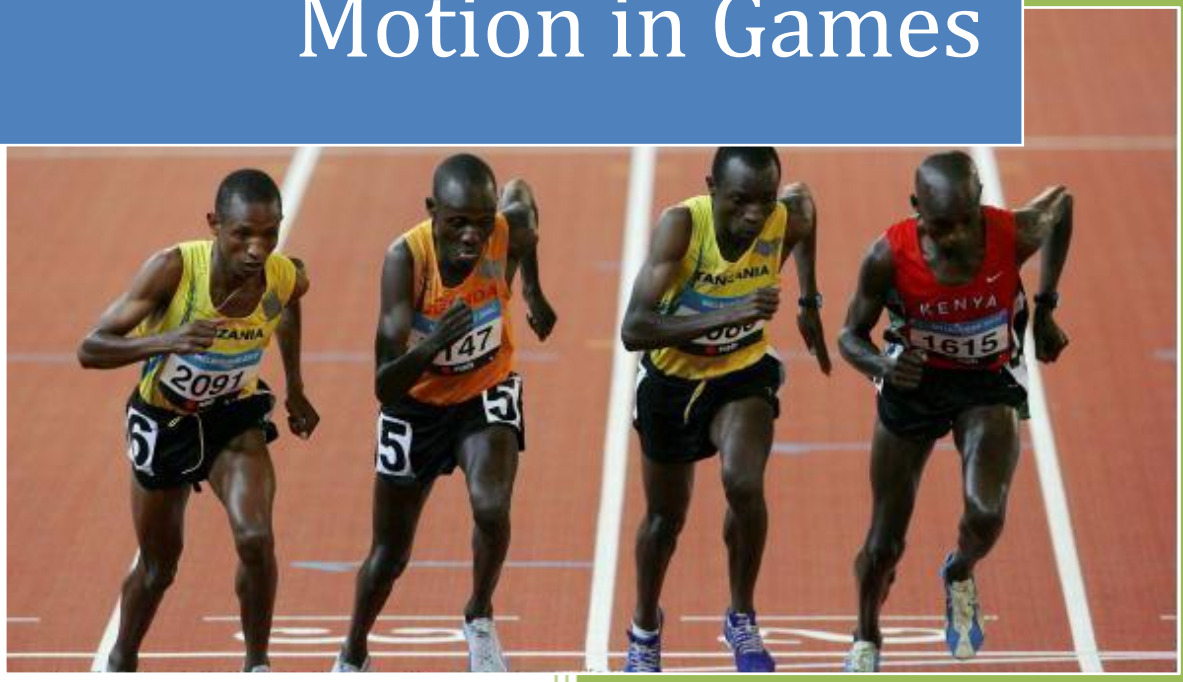
2008

Implementing Realistic Human Motion in Games



Elly Onesmo Nkya
ITU University of Copenhagen
Supervisor: Rune Møller Jensen
Msc. Thesis in Multimedia and Games

6/2/2008

Table of Contents

List of figures

Abstract

The computer games industry is increasingly being pressured to create games that are realistic, in the sense that rules that apply in the real physical world also do apply in computer games. Human motion (i.e. running and walking) is one area that is prevalent in computer games especially in genres such sports, action, and adventure. In real life, motion is a key element for all living beings if they are to engage. It is primarily governed by thought and actuated by muscles and their energy sources. Complex energy sources in muscles are used to produce force, which in turn creates motion. Currently, most games ignore, cheat, or implement rule-based logic to reflect the thought and muscle processes. In these cases, motion is not realistic.

This thesis attempts to model realistic human motion governed by realistic energy constraints, but can which realistically achieve long-term goals in a competitive real world environment.

An athletic event is used to construct a realistic competitive environment that allows us to focus on the energy constraints and capture the motion dynamics. Q-learning, a reinforcement learning method in machine learning, is then be used to model domain knowledge acquisition necessary for the decision making processes that produce realistic behavior for motion. Generalization of the behavior for larger environments using a combined Q-learning and neural networks model is then discussed implemented and measured.

Based on the theoretical analysis and the empirical results from the construction of the athletic event, this thesis concludes that a realistic model of the energy dynamics that control the way we execute motion be realized and implemented whilst also allowing them to generalize due to evidence the learning shows convergence. This model can then be used in computer games to increase the realism in chase and evasion strategies.

Acknowledgements

I would like to thank my supervisor, Rune M. Jensen for providing insightful feedback during the course of this work. Many of the methods and the material resources used to realize this work were introduced to me by him.

To friends, thank you for bearing with me as I went on and on about runners and energy. To Archard and Maureen, thanks for the review.

Rasa, thank you for all the care and tremendous patience while I worked on this thesis. To Christian and Anna, thank you for being such good children. To my mum and dad, you have been a constant source of encouragement.

Chapter 1. Introduction

"Games will become an immersive reality that mixes the game with the real world," Lambert said. "As the technology develops, we will see games combining the intellectual aspects of chess and the physical aspects of sports."

"A game should be an experience more intense than reality," Fan said. "For now, real life is still more fun than games."

(Crampton, 2005)

The Game Industry has been growing at a double digit rate. 90% of U.S. households with children have rented or owned a video or computer games, and young people in the United State spend an average of 20 minutes per day playing video games, making digital games the second most popular form of entertainment after TV. (Fullerton, Swain, & Hoffman, 2004)

1.1 Realism in games

With technological barriers being broken in hardware and multimedia capabilities, gamers have come to expect and demand more richness in the gaming experience. Aspects of games that were overlooked in the past are being called into question. Gamers now demand more realism. Realism is the implementation of real world behavior into games. This includes the mimicry of action and reaction of intelligence, sound, physics, lighting etc. Progress has been made. The evolution of 3D animated computer graphics has helped characters appear life-like (photorealistic). The evolution of higher sound quality has helped the gaming environment sound life-like and has improved interactivity. The evolution of faster processors has resulted in the development of engines that have helped the virtual world behave according to many laws of physics. The evolution of network has meant improved multiplayer capabilities enabling gamers to interact with intelligent beings virtually. However, these aspects have allowed gamers to take a closer look at non-player characters. And it has not been a pretty sight.

"In recent years the quality of graphics and sound has improved such that it is now easier to discern nonsensical or questionable actions on the part of non-player characters." "Computer Games With intelligence" – Daniel Jonson and Janet Wiles 2001

Life-like behavior of living beings in games is lagging, especially the non-player characters. AI could now be the dominant feature which makes a product stand out and become

competitive. Implementing realistic behavior is now gaining ground. It is present and noticeable in some of the best selling games. The bar of what can be presented as life-like behavior in games is therefore being raised.

1.2 Realistic human motion in current games

One aspect that remains unaddressed properly is realistic human motion.
Human motion is a basic part of many games. For the following genres it is essential.

- **Action games:** In this category are first-person shooter games which involve agent moving around as in real combat. Chasing and evading is a key part of the strategy in the game, under intense physical conditions of battle.
- **Sports games:** In this category lie games that simulate the experience of traditional sports. This is also an extremely popular genre (and includes some of the best selling games). Games include boxing, football, and rugby, which are physically demanding.

However, in all game genres little regard is given toward implementing motion as it appears in reality. This is can attributed to the following reasons.

- Biochemical processes that manage motion are complicated and hence are not implemented
- The thought process for motion is implemented using weak AI because of the assumed intractability of the environment. These methods, typically a combination of fuzzy logic and rule-based A.I. can cause problems. They can lead to poor behavior because they either do not take into account all the percept information necessary for rational behavior, or if they do, they compromise reliability of the program and make it hard to maintain.

This results in several issues:

- During a chase or evasion during chase and evasion the only strategy seems to be direction. There seems to be no other of strategy for an exit criteria or high order intelligence.
- Acceleration in player characters appears non-existent. All characters appear to be running at the same constant speed whether fresh from a chase or otherwise.
- Getting tired is almost non-existent even if a highly intense muscular activity has just been performed. Chasing or evasion can continue infinitely unless it chooses to enter

one of those nonsensical trances of defeat. With the absence of constraints non-player character and player characters expend their resources recklessly.

1.3 Motion in the real world

Motion or locomotion in human beings is an important aspect of everyday life. Walking, running, jumping, crawling, climbing, and swimming are some of the types of motion one can exhibit. The ability to perform motion effectively allows living beings to achieve their goals.

It involves aspects of physics, biology, chemistry, and decision-making to manage optimal motion

In the real world, motion is achieved by exerting force using skeletal muscles on the limbs. This exertion of force produces a resultant acceleration/deceleration. Force can only brought about if there is enough chemical energy to be converted to kinetic energy. There are several types of chemical energy sources. Each source has its own storage limits. Conversion rates to kinetic energy depend on the energy store and on the type of chemical reaction (aerobically or anaerobically) necessary for conversion. Aerobic respiration, which sustains longer intense motion, is also more effective when the human is moving slower.

Therefore, to maximize the outcome of a motion activity, humans must balance acceleration with this complex energy production. Given enough experiences they can optimize the right balance between the two. The thought processes is therefore important in achieving optimal results.

1.4 Advantages of including realistic motion in games

There are several advantages that will be achieved from implementing realistic human motion in games.

- Engages players to be more strategic with their resources
  Take tennis for example: making your opponent run around the court is part of the strategy of tiring him.
- Encourages cooperation in multiplayer games
  Passing the ball around in football is a key strategy in keeping players from getting tired instead of going at it alone no matter how good you are.
- Potential for opening up possibilities for more physical endurance games

Many physical endurance sports are yet to be implemented or are unsuccessful. The addition of this key element may spark more interest in having athletic running events, rugby, tennis etc.

- Improves the perception of life-like behavior in non-player human characters
- Can act as a simulation to be applied to endurance sports training
- Strong A.I. can end-up creating non-player characters that do not need to resort to cheating in order to compensate for their incompetence. Fairness can is a key component of game play. It also may open the possibility of non-player having the capability to adapt to the competition and hence make the game more engaging.
- The cost of maintaining the system is minimized if strong A.I. methods are used to mimic behavior. This is because of their resilience to error. Developers therefore do not need time bug fixing their way to realistic behavior.

1.5 Thesis question

In this thesis, we will attempt to model realistic human motion by incorporating all the basic real-world dynamics that underlie human motion. This will include modeling of the muscles energy sources, respiration, and conversion of the chemical energy to kinetic energy. The effects of the forces that result from the kinetic energy production will also be discussed. Realistic perception and sensation, which plays a key part in how humans acquire relevant information about state of the environment to which the task of motion is required, will also, be modeled.

Decision-making plays a key role, if a human is to achieve a goal that requires motion. Today, no domain theory exists on how the brain makes decisions on what force the muscles need to produce for motion, given the complex energy sources. But we do see that the brain does make choices that are rational and is capable of achieving long-term goals requiring motion optimally. Machine learning methods are the closest we get to mimicking realistic human intelligence, among them reinforcement learning is capable of producing domain knowledge through experience. However, it cannot support dynamic environments with large state-spaces. Artificial neural networks, are capable of generalizing domain knowledge once it is known, by expressing it in a compact form that can support environment such as those for motion. The question for this thesis then becomes:

“Is it possible to create realistic motion for human characters in games, based on a realistic energy models using a combined model of reinforcement learning and neural networks?”

1.6 Choice of environment to study realistic motion

To explore the domain of motion in humans we need also choose a “real-life” environment that allows us to focus on the problem. Normal everyday life environments are too complicated to use for learning motion, due the physical configurations, and many secondary goals involved.

Athletics is a competition of human physical abilities. These events are usually held in well defined environments governed by simple rules that allow the athletic abilities to the central focus. The relative simplicity makes it possible for us to focus strongly on learning human motion. We will focus especially on the running events of athletics.

1.7 Scope

There are numerous factors that impact motion. In this thesis we will focus on key factors, These will be include, muscles energy sources, chemical energy conversion, respiration, the competition, the actual force that produces movement expressed as velocity and position. Sensation and perception of these is also important for realistic decision making. Memory or prior knowledge plays a big role if agents are to learn from their experiences, which is typically of human behavior.

There are also several other factors which will not be addressed in this work. To limit the complexity, we will assume negligible effect on these aspects. They include both internal and external factors.

Internal factors: are those emanating from the human himself. Limb movement can be effective in optimizing the energy sources and reducing stress to the muscles. Posture can also reduce the effect of weight. Diet is known to also control weight, contribute to energy sources, and contribute to heath that is vital to the development of organs used in running. Exercise and practice also contribute in appropriate muscle growth, development of the lungs for oxygen intake, and better running strategy.

External factors: are those factors that emanate from the environment: Wind depending on the direction can inhibit or assist motion. Humidity can cause the runner to spend too much energy ridding the body of heat. Altitude contributes to pressure change that adds to another force against motion.

1.8 Thesis contributions

The contributions of this thesis are twofold:

1. It proposes a solution concept that can be used to model realistic on human games characters based on realistic sensation and perception, realistic energy sources, realistic movement, and decision-making processes mimicking rational thought.
2. Implements and evaluates the performance of such a model based on an athletic competition and presents workarounds, shortcoming, and ideas for further improvements.

1.9 Structure of this thesis

The remainder of the report is organized as follows. In chapter 2, we will introduce the background of the key concepts, which we will build this thesis upon. We will begin by an introduction to current game architecture and its basic parts. This will provide the context of the design constraints to which we model motion upon. This chapter will continue with an introduction on the physical and chemical aspects that brings about real-life motion, including the energy sources, respiration, breathing, and actual movement. An introduction on how decision-making processes necessary for motion will be presented and we will by introducing Q-learning and neural network. These methods of machine learning will later be used to mimic human motion decision-making processes.

In chapter 3, we then address the question. We will start off by modeling a realistic athletic competition into the game architecture. Following that, we will model realistic motion actuation which captures breathing, energy conversion, and kinematics. Models for sensation and perception for realistic human will then be realized. This will also help in identifying the characteristics of the environment in A.I. terms, which we later use for modeling decision-making. The final part of this chapter will then address the modeling of realistic decision making processes for human motion. A combined model of Q-learning and neural network will be proposed and argued for.

In chapter 4, the empirical results will then be presented and discussed, focusing on key characteristics that address the thesis question.

In chapter 5, we will conclude our findings by providing a summary of what was achieved, the contributions of this thesis, and insights on future work that can address shortcomings that were identified during the course of this work.

Chapter 2. Background

Before we move on to describing how we can model realistic human motion, we will introduce several fundamental concepts that are the building block for this thesis.

In this chapter will begin by introducing the architecture of a typical modern game. We will focus mainly on the game logic framework. We do this because the modeling of human motion will be constrained under current architecture of game development.

The mechanisms that bring about real human motion will then be introduced. This includes energy conversion in muscles, breathing, and kinematics. This is a broad subject involving a lot of complexity and unknowns. Hence the description will only focus on aspects that are necessary to solve the thesis problem. This may also include the authors' assumptions in gray areas.

A brief introduction on athletics will be made as we will base the game environment on an athletic event. Key aspects of the competition and its rules will be introduced.

Decision making forms the central part of this thesis; hence the later sections will briefly introduce realistic thought processes. AI methods, specifically Q-learning and Neural network, which are used to model learning, thought and decision making, will then be introduced.

2.1 Modeling the game

Computer games architecture usually features three basic parts:

- Initialization: deals with the loading of the game graphics, initialization of the game settings. Games setting usually consist of the default settings, game rules, domain knowledge, location of game graphics etc. Of key interest to our thesis, the environment and agents involved are primarily loaded here as well. Lines 6-7 on the algorithm presented below describe this.
- Game Episode

  This is the central part of the game. It manages the game plays until the criteria for ending is encountered (Line 15). The episode includes a game loop which is an important part in handling animations in dynamic environment where refresh of the graphics is important. Lines 9-16 of the algorithm represent the game episode.

  Another use for it is in managing the agents that interact with the environment. Agent behavior can be divided in three parts, sensing the environment, decision-making, and actuating on the environment.
- Finalization

  This is the final part of the game. It may include release of memory, saving updated games setting, and the presentation of results. See Line 16-17.

The following algorithm presents an overview of the Game Episode algorithm.

```
1   function GameEpisode(gameSettings)
2     inputs: gameSettings contain the default settings of the game including the
                  domain knowledge, graphic inputs, and game rules
3     local variables: environment,
4                      agents, who will be competing in the game
5
6   InitialiseGame
7     environment ← InitializeEnvironment(gameSettings)
8     agents ←InitializeAgents(gameSettings)
9   GameEpisode
10    repeat
11      foreach a in agents
12        percept ← Sense(a, environment)
13        actions ← Decide(a, percept, [domainKnowledge])
```

```
14            environment ← Act(a, actions, environment)
15        until GameEpisodeStoppingCriteria(environment)
16    FinalizeGame
17        PresentResult(agents)
```

Table 2.1-1: The Game Algorithm

2.2 Producing motion

The movement of the center-of-mass is what is termed as motion. Motion is achieved by sophisticated skeletal muscle contractions and dilations on the limbs. These muscles movements cause the limbs to rotate about the joints.

2.2.1 Force and motion

The limb movements cause the feet to exert a force against the ground that causes the body to move forwards. Force consist of a magnitude and direction, and are expressed as vectors. Friction and atmospheric drag work against this force created by the muscles. Motion occurs when the muscles force exceeds that of the opposing forces, the center-of-mass of the human body shifts from one position to another. The relation between the forces can be represented by the following relation,

$$F_{resultant} = F_{muscles} - F_{drag} - F_{friction}$$

Equation 2.2-1 The resultant force of motion

Where

$F_{muscles}$ is the force vector produced by the muscles,

F_{drag} is the opposing force vector caused by the atmosphere due to the resistance of air. (Drag (physics)) describes on the underlying nature of this force,

$F_{friction}$ is the opposing force vector between brough about as result of the contact between ground and the feet. (Friction) describes the nature of this force.

2.2.2 Energy storage and usage

Muscle activity requires energy. To produce energy destructive metabolism commonly known as catabolism should occur. Catabolism is the process that produces energy required for all activity in cells. In this process, large molecules cells (mostly carbohydrates or fat) break down to release chemical energy. This energy release provides for among other things, kinetic energy used to produce motion through muscles contraction. During this conversion, some

energy is lost as heat due to chemical activity and friction. In fact, only a ¼ of the energy is turned into useful power (Body Atlas, 2006). Waste (heat and carbon dioxide) by products of energy production is released into the blood which flows back to the heart. The conservation of energy can be used to describe the relation between the energy as:

$$E_{chemical} \rightarrow E_{kinetic} + E_{heat}$$

Equation 2.2-2 Energy conversion for motion

Where,

$E_{chemical}$, is the chemical energy of the muscles energy sources,

$E_{kinetic}$ is the kinetic energy produces as a result of the chemical conversions.

E_{heat} is the energy lost as heat during the chemical conversions to kinetic energy.

The body stores energy in various forms. However, muscles energy is mainly derived from Fat, Glycogen, Glucose, or ADP. To produce energy, catabolic activity has to convert them to ATP. ATP is the chemical compound that converts easily to kinetic energy.

Each energy source has its own properties.

- Fat, is a form of energy, which functions as a long term energy storage. It is in abundant supply and can sustain muscular activity with low intensity for a long periods of time. Before it can be converted to ATP, it is first converted to glucose. Its conversion rate to glucose is too slow for intense activities.
- Glycogen, is a form of simple sugar which functions as a short term energy storage. It can readily be converted to glucose. However it is in limited supply and hence can be exhausted after a short period of intense muscle activity.
- Glucose is another form of sugar which serves as an intermediate form between energy stores and ATP. When the body discovers that it has more than can be used immediately, it is converted back to glycogen. Glucose can be converted aerobically or anaerobically to ATP, depending on the availability of oxygen. Studies show that anaerobic conversion of glucose can only be sustained only for a very few minutes. Exhaustion of this can cause muscle pain due to rising levels of lactic acid.
  Aerobic conversion can sustain muscles activity for a long time; however it is highly dependent on the oxygen-rich blood being pumped to the muscles.

- ADP is an anaerobic source of energy. In the event that glycogen cannot meet the demand of energy required at the rate required aerobically, this will be used. Alone, it can only sustain 30 seconds of intense muscles activity.

The brain may cease all voluntary muscles activity, if the level of carbon dioxide in the blood is detected to reach a certain threshold. This would cease motion, keeping the heart pumping only for life threatening activities. This state is commonly known as a blackout.

Below is a table that describes the various characteristics of each the energy source.

Primary Sources of Muscle Energy	**Conversion to ATP** (note the equations below are not balanced. They also exclude by-products such as heat energy)	**Conversion Rate**	**Anaerobic/ Aerobic**	**Availability**
Fat	$Fat \rightarrow Glucose \rightarrow (see\ glucose)$	Very Slow		Abundant
Glycogen	$Glycogen \rightarrow Glucose \rightarrow (see\ glucose)$	Slow		Moderate. Can sustain 10km
Glucose	$Glucose + Oxygen + ADP + P_i \rightarrow ATP + CO_2 + H_2O$	Moderate	Aerobic	Produced from fat or glycogen. Depends on the availability of O_2
	$Glucose \rightarrow ATP + Lactic\ Acid$	Fast	Anaerobic	Low. Can only sustain 90 seconds of intense muscle activity
ADP	$ADP + P_i \rightarrow ATP$	Very Fast	Anaerobic	Low. Can only sustain 30 seconds of intense

				muscle activity

Table 2.2-1: Muscles energy sources conversion

2.2.3 Aerobic respiration

Aerobic respiration is therefore critical for muscle activity that lasts for longer periods. To be able to supply the oxygen that is needed by these muscles, humans need to inhale air, which is then transported by the blood via the lungs to the muscles. Inhalation varies depending among other factors, the speed at which the runner is at. Oxygen inhaled is inversely proportional to the speed of the runner. Hence as a runner attains higher speeds through acceleration, muscles activity increases which implies energy consumption whilst oxygen intake decreases. If unchecked the runner will collapse.

The heart also plays a central role in ensuring as much oxygen arrives to the muscles. This is achieved by increasing its rate of pumping blood.

Below is a diagram that gives an overview of the process of aerobic respiration that is actually described in (Body Atlas, 2006).

Table 2.2-2: Aerobic respiration cycle

.

For more on muscle energy use see (Horn), (Body Atlas, 2006). and (Adenosine triphosphate).

2.3 Decision making

All decision making processes in humans occur in the brain. This central organ is aided by both short and long-term memory capabilities. So with constant repetition of training, a human can learn to understand his competition, understand his limitations, and also make best use of his body and lungs in order to be successful. The heart among other things also controls the breathing rate, the heart beat rate all of which are central to motion. Carbon dioxide triggers an alarm in the brain automatically stimulating nerves that control the chest muscles and diaphragm. For a short time you can consciously override the breathing reflex but it is impossible to suffocate yourself. The heart also controls the amount of oxygen in the blood by detecting carbon dioxide levels. Decisions from/to the brain are done through nerves using the spinal cord as a passage. The follow are some of the motion decision-making process the brain controls summarized from (Body Atlas, 2006).

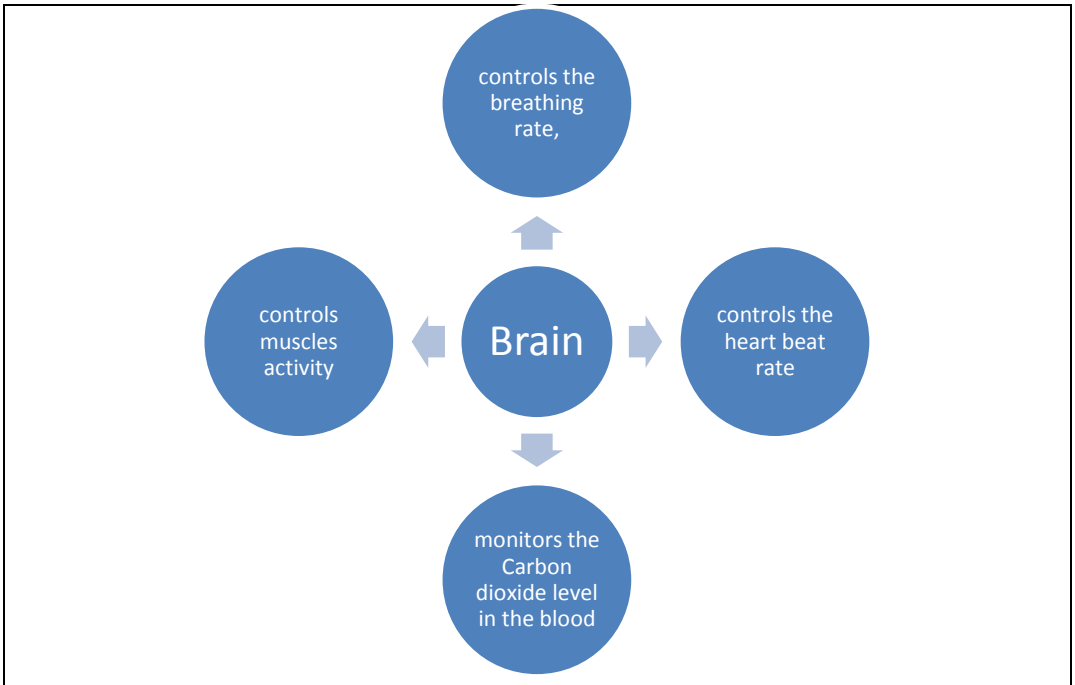


Table 2.3-1: The brains role in motion

Given the energy constraint described above, human needs to make good decisions in order achieve goals that require motion. In the long distance running for instance, running at full speed would not be wise if one is to complete a race. Chasing scenarios which are common in games would require decision making on how to use energy optimally before and during the chase if one is maximize the outcome of the activity. This is because the environment is in continuous change also because the object being pursued be it in motion is also changing and has strategies that may be unknown to the pursuer. This goes for evasion strategies as well.

2.3.1 Q-learning

Q-learning is a reinforcement learning method that allows agents with no prior domain knowledge in a sequential problem how to learn to choose actions that maximize the outcome of a long-term goal following a series of experiences.

The Q-learning framework consists of the environment, sensor, actions, and rewards.

- Agent: the entity the needs to learn. In case of the runners, the agent would be that part of the brain that is responsible for motion decision-making.
- Environment: consist of all entities that are affected by the action that the agent takes. For example, in case of the runners, the energy stores, the position of the body with respect to the running track can be considered part of the environment.
- Sensors: the means in which the agent is presented an interpretation of the environment necessary for him to choose actions. For example, in running, sight is a sensor used to interpret our position with respect to the finish line.
- Action is response the agent make that causes the environment to change state. For example, in the case of runners, choosing to accelerate causes the body move.
- Reward is a numerical response from the environment to the agent that reinforces the action previously performed.

The diagram below gives an overview on how the components are related at any time step t. Here agent senses the state, $s(t)$ of the environment, and responds by taking an action, $a(t)$ chosen among several possible ones in that state. The action causes the environment to change to a new state. The new state, $s(t+1)$ is then sensed by the agent together with a reward, $R(t+1)$. The agent's goal is therefore to maximize the cumulative reward, $R(t+1) + R(t+2) + R(t+3)$ ... that it will receive by performing these actions, this in effect will also allow the agent to achieve a long term goal optimally.

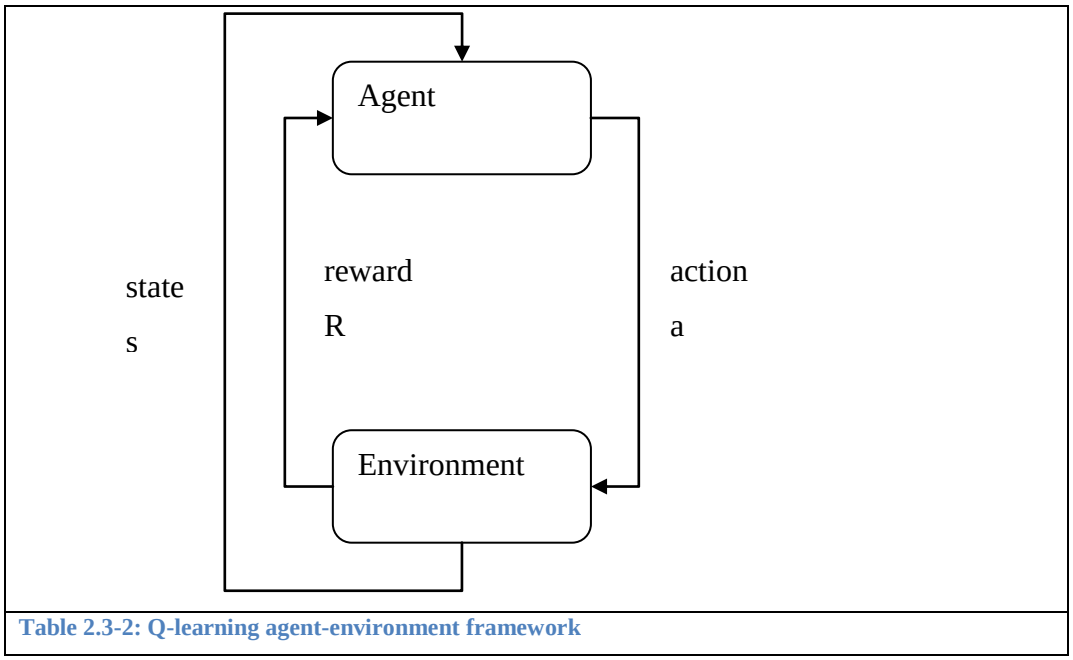


Table 2.3-2: Q-learning agent-environment framework

The Q-learning agent program produces this optimal behavior by associating a numerical value, known as the Q-value to each state-action pair it has visited. The Q-value represents the desirability of choosing an action among several possible choices in that state. Q-values with higher values are more desirable.

These Q-values are learned through the agent interacting with the environment and receiving rewards. The values are initially inaccurate, but with more interaction with the environment they get to be accurate and hence the agent becomes more consistent in achieving the goal. Q-values can be updated as follows:

$$Q(a,s) = Q(a,s) + \alpha\,(R(s) + \gamma\, \max_{a'} Q(a',s') - Q(a,s))$$

Equation 2.3-1 Q-value update for a non-deterministic MDP

where,

$Q(a,s)$ is the Q-value of action a in state s,

α is the learning rate,

$R(s)$ is the reward at state s,

γ is the discount factor,

$Q(a',s')$ is the state that follows when action a was performed in state s

The formula, above works by slightly improving the previous Q-values $Q(a, s)$, of the previous state s, using the reward that was sensed $R(s)$, and the maximum Q-value that can be obtained by choosing an action in the current state s'. The rate of improvement is controlled by the learning rate, α.

This specific update method presented above is relevant especially for environments that are stochastic in nature, meaning, there is uncertainty in the state will be produced in the next time step as a result of performing an action in the current state.

The Q-learning agent program is based on the assumption that the state percept possesses the Markov Property. The Markov property holds when the current state percept (sometime including a finite state history) contains enough information to predict the future without having to inspect the entire percept history.

For further reading about this method please refer to (Sutton S & Barto G), (Russel & Norvig, 2003) and, (Mitchell, 1997).

2.3.2 Artificial neural networks

Artificial neural networks are known for their powerful capability of learning to classify statistical probabilistic patterns (Bishop C. M., 2006). The networks consist of neuron (nodes) that are interconnected by directed edges that carry a weight. The networks can be constructed in various configurations, to suit the problem, but in essence consist of three essential parts; the input layer, an optional hidden layer, and output layer. The networks form a mathematical mapping of the form

$$f(x) \rightarrow t$$

Where,

$x \equiv (x_1, x_2, \dots, x_N)^T$ are the input signals vector accepted through the input layer which contains N nodes, each node accept a value x_i and,

$t \equiv (t_1, t_2, \dots, t_M)^T$ are output signals that are output through the output layer containing M nodes, each node represent a value t_j.

The mapping f, is performed by rippling the inputs through the network using the directed edges and strengthening them by their weights, until they arrive at the output node. The

signals arriving at each node in the hidden layer or output layer, are activated, using an activation function in the form shown below,

$$a_j = g\left(\sum_{i=0}^{d} w_{ji}\, x_i\right)$$

Equation 2.3-2 Neuron activation function

Where,

a_j is the output signal of neuron j,

g is the activation function.

w_{ji} is the weight of on the edge ji,

x_i is the input signal

To represent a non-linear mapping, the activation function needs to be non-linear as well. Common activation functions used are the logistic and sigmoid functions, which are continuous, non-linear, and differentiable. The differentiable characteristic allows us to minimize the error during training of the network.

A neural network can also perform the task of learning the mapping relation f, between x and t. The task of a learning neural network is of the following nature; to take examples comprising of N observations of inputs x, together with the corresponding observations of the output value t, and to produce a network also that map these examples with minimal error. Most of the learning is done by adapting the weights w_{ji} so that the network produces correct outputs. The number of hidden nodes in a neural network plays a large in the networks capability to learn highly non-linear problems. But too many hidden nodes can also affect the performance of network to classify unseen examples.

In this thesis we shall particular attention to one particular configuration, an acyclic, fully connected (only between nodes in adjacent layers) feed forward network. In fact, studies show that it is possible to represent any continuous functional mapping, using a two-layered network to some degree of accuracy, using a sufficient number of hidden nodes. The example below, serves to illustrate its essential features.

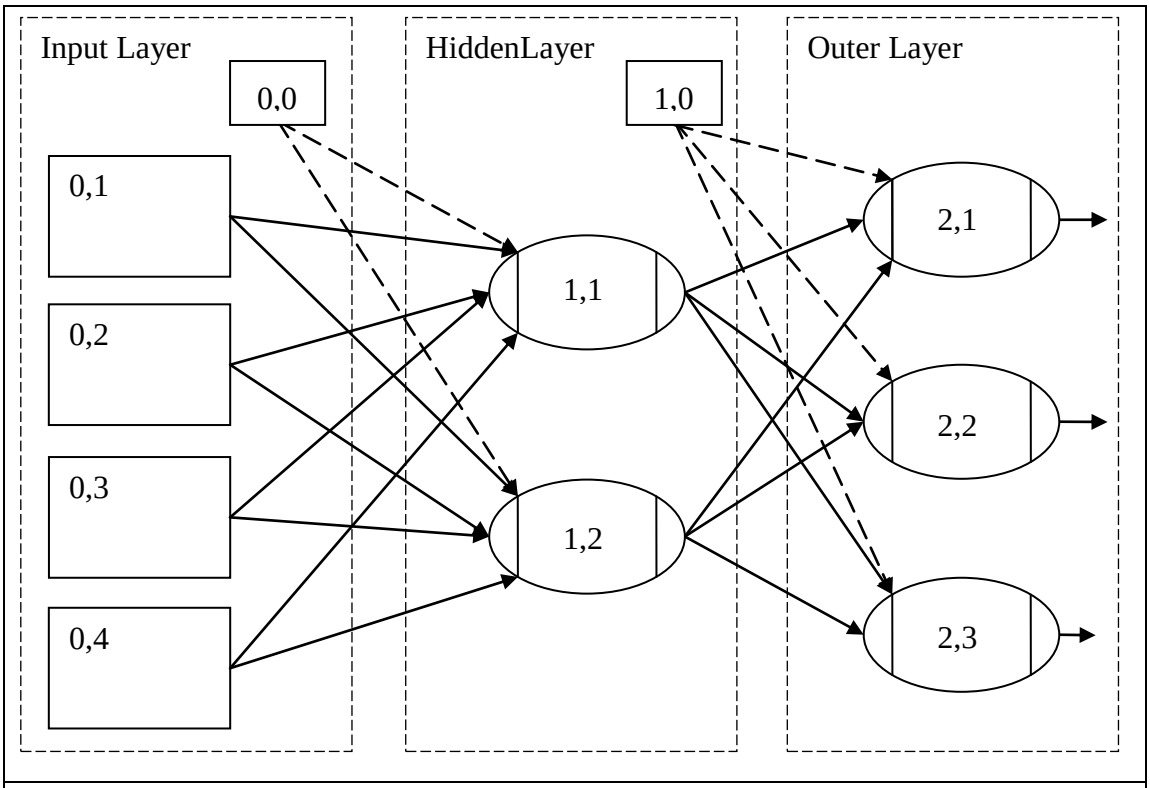


Table 2.3-3: An acyclic, fully connected, feed forward multilayer neural network example

The network is acyclic because no edges flow backwards

It is fully connected because each node receives signal from all other node in the preceding layer

It is a feed forward network because the signals are directed to only flow forwards

The oval shapes represent neurons divided into three parts to represent the accept of signals, activations of the sum signal and output. The network above is a two-layer network, because only two contain neurons that can be activated.

The small box represent bias nodes usually possess a fixed input value and weight that are adapted during training.

2.3.3 Roll-up of a neural network

In an acyclic, fully connected feed-forward network, the activation in any node $[l, i]$ is computed using following recurrence relation. Activating the outer layer node causes the hidden layer and input layer nodes to also be activated. This is result in a roll-up of values through the network.

$$activ(x,l,i) = \begin{cases} g\left(\sum_{j=0}^{n} activ(x,l-1,j)w([l-1,j],[l,i])\right) & l,i > 0 \\ -1 & i = 0 \\ x[i] & l = 0 \end{cases}$$

Equation 2.3-3 Rollup activation function

where,

g is the **activation function**. Activation functions typically used are the sigmoid or the tangent hyperbolic functions.

w: (represented as vertices) are the **weights** usually represent a vertices,

Nodes $activ(x,l,0) = -1$, in the outer and hidden layers are known as the **bias**.

Nodes $activ(0,i) = x[i]$ represent the value of an input vector attribute

2.3.4 Learning using backpropagation

Neural network learning is performed by adjusting the weights w of the network. One effective method used to do this is known as backprogation. What backpropagation does is it adjust the weight by rippling the share of the error $\Delta(a,l,i)$, which typically uses the intuition of the mean square of error which is the half the square of the difference between the expected output vector t and the actual value obtained from a roll-up $activ(a,l,i)$. Mean square of error allow for the error to minimized using the gradient descent method. Below is how the weight update is computed.

$$w([l-1,j],[l,i]) = w([l-1,j],[l,i]) + \alpha \times activ(a,l-1,j) \times \Delta(a,l,i)$$

Equation 2.3-4 Weight update in neural network learning

and

$$\Delta(a,l,i) = \begin{Bmatrix} g'\left(activ(x,l,i)\right) \times (t[i] - activ(x,l,i)) & l\ is\ an\ outer\ layer \\ g'\left(activ(x,l,i)\right)\sum_{j=0}^{n} \Delta(x,l+1,j)w([l,i],[l+1,j]) & l\ is\ a\ hidden\ layer \end{Bmatrix}$$

Equation 2.3-5 Backpropation error in neural network weight update

Where,

α: is the learning rate

g' (): is the derivative of the activation function

Example

We use the network presented in Table 2.3-3: An acyclic, fully connected, feed forward multilayer neural network example. let $l = 2, j = 2, i = 3$,

Therefore,

$$w([1,2],[2,3]) = w([1,2],[2,3]) + \alpha \times activ(x,1,2) \times \Delta(x,2,3)$$

Where

$$\Delta(a,2,3) = g'\big(activ(x,2,3)\big) \times (t[3] - activ(x,2,3)$$

For further details on neural network can be found in (Bishop, 2007) and (Mitchell, 1997)

Chapter 3. A game motion agent combining Q-learning and neural networks

In this section, we will go about modeling an agent program that addresses the thesis question.

We will begin by presenting a realistic model of an athletic game architecture and layout it most prominent features, whilst staying true to the classic game architecture presented earlier in section 2.1 .

Following that, we shall propose a model that reflects realistic actuation for human motion. Here we eventually end up with an algorithmic design that captures breathing, energy conversion, and kinematics.

Modeling of sensation and perception for human motion will be addressed. This analysis will allow us to capture the complexity of the environment that will aid us in identify the most appropriate agent program for the modeling motion decision-making processes. Here we will also propose the algorithmic design for sensation and perception.

Finally, but most importantly, the thesis will then model the actual decision-making process that produces realistic motion. This is subdivided into two parts, direction and the magnitude of acceleration. The athletic competition will first be fully characterized in A.I. terms. Following that, reinforcement learning will be introduced as method for domain knowledge acquisition. The generalization of the domain knowledge then be addressed using a combined model featuring Q-learning and neural networks. Complete algorithms capturing the decision making for realistic motion will also be proposed.

3.1 Modeling the athletic competition for realistic motion

In this section we relate a realistic athletic competition to the classic game architecture introduced in section 2.1. Specifically we take an Olympic running event and relate it to the initialization, episode, game loop, and finalization. In so doing, we demonstrate that realistic motion can actually be fitted easily in current game architecture without major overhauls.

3.1.1 Initialization

Initialization in an athletic running competition involves several aspects that are of interest to motion,

- Loading of the running track and the rules of the competition described by (Competitions - Rules and Regulations).
- Loading the runners who will participate, random initial positioning of them on the track at the start line with their initial energy configuration, and prior knowledge of motion. This initial state should not create any unfair advantage to any athlete. But for the sake of observing variations in behavior we will assume some slight randomization in the initial value of position and energy sources. Lines 6-9 in the algorithm below describes this part of the game.

3.1.2 Episode

The actual Olympic race easily maps to the computer game concept of an episode. Each athlete is required to run around a track whilst following rules of the competition until a stopping criterion is met. One stopping criteria that can be used is once every runner has finished the race, that is, either by crossing the finish line or being disqualified. In an episode, each runner will be required to perform a sequence of actions, mainly to produce motion. Such an environment is defined as **sequential** in AI terms. This sequence can be controlled using the game loop, between lines 10-16.

3.1.3 Game loop

This part includes sensing, perceiving, and acting logic of each runner that is participating in the race. It is also where the game graphics are managed. For a game that will require animations such as this one, it should at least be processed at least 24 times per second for frames refresh (David Hearn, 2004). So spending too much computation time on game logic is undesirable. Hence realistic human motion logic should not require too much

computational resources. For fairness, it may be observed that each runner is allowed a chance to act on each game loop is he is still in the race.

At this point, we can already start replacing abstract terms used in Table 2.1-1: The Game Algorithm with more specifics.

The algorithm below describes the analysis .

```
1   function AthleticsGameEpisode(gameSettings)
2    inputs:  gameSettings
3    local variables: track, contains the athletic environment and its competition
                        rules
4                       runners, the set of runner competing in the race each with a
                          velocity vector v, position vector p, and energy store e,
                          and race status rs
5                       ∂T, the time step that elapsed since the previous game loop
6   InitialiseAthleticEpisode
7     track ← InitializeRunningTrack(gameSettings)
8     runners ← AddRunnersRandomlyOnToTrack(gameSettings, track)
9   AthleticEpisode
10    Repeat
11      foreach r in runners
12        if rs[r]is InRace or NotStarted then
13          percept ← Sense(r, track, runners)
14          acceleration,direction←DecideAction(r, percept, domainKnowledge)
15          v[runner],v[position],r[eStore] ←
              ActuateMotion(r, acceleration, direction, ∂T)
16    until RunningEpisodeStoppingCriteria(runners, track)
17  FinalizeAthleticEpisode
18    PresentResults(runners)
```

Table 3.1-1: The Athletics Competition Algorithm

It is no longer necessary maintain a runner in the game loop once he has completed the race, the primary action as line 12,

The primary action runner should be making decisions upon is acceleration, split into its direction and magnitude component (line 14).

The actuation of motion should at least produce a new state in velocity, position, and energy source for the runner (line 15).

3.2 Modeling actuators for realistic motion

In this section we describe how we model actuation of motion in an athletic running competition. We address the 3 key components that bring about motion.

- Breathing, which is necessary for aerobic respiration, will first be modeled, and an algorithm proposed.
- Actual motion, which involves moving the center of mass from one point to another. Rigid body kinematics will be addressing this.
- Lastly, energy conversions, which bring about the possibility of motion, that were introduced in section 2.2.2 will then be modeled and an algorithm proposed.

3.2.1 Breathing

Oxygen is a key component in the catabolic activities that convert glucose to ATP, which powers the muscles to create motion. Section 2.2, presented a detailed account of aerobic respiration, which led to the insight that breathing the provider of oxygen was an essential part of motion. Oxygen that arrived to the muscles was also affected by the speed of the runner, because breathing becomes harder when in motion. Below, we present an algorithm that computes the amount of oxygen available to the muscles. We shall also assume a complex relation non-linear relation on the amount available in this form.

$$oxygen\ available\ to\ the\ muscles\ \propto \frac{1}{current\ speed\ of\ the\ runner^2}$$

Equation 3.2-1 Oxygen available of muscles

```
1  function Breathe(speed, ∂T)
2    returns new oxygen in blood available to muscles of runner
3    inputs: speed, the current speed of the runner
4            ∂T, the time step that elapsed since the previous inhalation
5    statics: ∝, the rate of breathing oxygen
6
7    o2 ← ∝ ∂T / (1 + speed 2)
8  return o2
```

Table 3.2-1: Runner.Breathe Algorithm

In the actual implementation, the breathing rate was chosen, such that it could support running at 5m/s using only aerobic respiration of available glucose. Increasing speeds, would then rely on anaerobic means to provide for the energy demand.

3.2.2 Rigid body motion

We stated earlier, that human motion consists of a series of complex angular movements by the limb skeletal muscles about their joints. To compute exactly what goes on is complicated and deserves a thesis in its own right, hence beyond the scope of this work. However, for the sake of simplicity, we will assume the resultant forces created, equals that of rigid body motion. After all, we are interested in the movement of the center of gravity from one point to another. Rigid body kinematics deals with exactly this kind of motion and contains solid theory based on Newtonian mechanics.

We expressed the resultant force for human motion based on the following relation in Equation 2.2-1 The resultant force of motion

$$F_{resultant} = F_{muscles} - F_{drag} - F_{friction}$$

Where,

$F_{muscles}$ is the force created by the muscles, primarily governed by instruction from the brain to accelerate by ā. Hence, $F_{muscles} = m\bar{a}$, assuming the human body is a rigid body.

F_{drag} is the opposing force created due to atmospheric resistance. i.e. wind. These are most quite negligible in most conditions. However we can state that, $F_{drag} \propto \bar{v}$. Hence, $F_{drag} \propto k\bar{v}$. During an athletic competition, weather conditions are usually not expected to change radically in the course of the event hence we shall assume$k = c$, where c is a fixed constant for the sake of simplicity.

$F_{friction}$ is the friction created by the feet contacting the ground. Running on ice usually explains the significance of friction. $F_{drag} = \mu F_{Normal} = \mu W = \mu mg$, Where μ is the coefficient of friction and W is the weight of the solid (runner). In normal athletic conditions, the effect is usually not extreme and does vary during the time of the event because the track surface is uniformly of the same material, hence we assume $\mu = k$, where k is a fixed constant for the sake of simplicity.

This gives rise to the following algorithm, that allow us to compute the new velocity, position, and energy required, for a runner to move in a given time step ∂T. Line 12-13 compute the

resultant force of the motion based taking the minimal effects of drag and friction. Lines 15-17, computes the change in position of the runner, and the new velocity achieved. Line 18, computes the energy required to move the runner to the new position.

1	**function** Move**(***runner***,** a, $\bar{d}$, ∂T**)**
2	**returns** new velocity of runner, new position of runner, kinetic energy required
3	**input:** a, the new magnitude of the acceleration
4	$\bar{d}$, the new direction unit vector of the runner
5	∂T, the time step
6	*runner*, one competing in the race containing his current velocity vector $\bar{\boldsymbol{u}}$, and position vector $\bar{\boldsymbol{p}}$,
7	**local variables**
8	$\bar{v}$**,** the new velocity vector of the runner.
9	$E_{required}$, the energy required to produce motion
10	$\bar{p'}$, the new position vector of the runner
11	
12	$\bar{a} \leftarrow a \times \bar{d}$
13	$\bar{a}_{resultant} \leftarrow \frac{m\bar{a}-k\bar{u}-\mu mg\bar{d}}{m}$
14	
15	$\bar{v} \leftarrow \bar{u} + \bar{a}_{resultant} \times \partial T$
16	$\partial\bar{p} \leftarrow \bar{u} \times \partial T + \frac{1}{2}\bar{a}_{result\ ant} \times \partial T^2$
17	$\bar{\bar{p'}} \leftarrow \bar{p} + \partial\bar{p}$
18	$E_{required} \leftarrow m\bar{a}\partial\bar{p}$
19	**return** $\bar{v}$, $\bar{p'}$, $E_{required}$

Table 3.2-2: Runner.Move Algorithm

Produces linear motion using rigid body kinematics. This algorithm assumes the runner behave like a solid object with no rotational motion.

3.2.3 Energy conversion

The preceding two sections in motion actuation provided the necessary ingredients for energy conversions, i.e the kinetic energy required E_k, and O_2 available in the time step ∂T. Here we model how the fulfillment of the E_k, the energy required for motion is done. The theory on this work was presented section 2.2.2.

The algorithm below outlines the basic essentials energy conversion that the body muscles undergo in order to fulfill the energy request. It first converts glycogen and fat to glucose line

7-9. Fat is only converted to glucose if glycogen already is below a storage limit. Any excess glucose that is unused is converted back to glycogen, see lines 16-17. Following the conversion to glycogen, production of ATP ensues in the sequence of aerobic-glucose, adp-anaerobic, and lastly anaerobic-glucose, see lines 10-15. Atp is requested from the sources, only if the preceding sources cannot fulfill the requirement.

```
1   function ProduceKineticEnergy(eStore, O2, Ek, ∂T)
2    returns new energyStore of runner
3    inputs: O2: oxygen in blood available to muscles of runner
4            Ek : kinetic energy required by the muscles
5            ∂T : the time step´
6            eStore: the set of energy stores of fat, glucose, glycogen, adp, atp, and
                 the the efficiency eff representing the amount of  lactic acid
7   glucose ←ProduceGlucose(glycogen, ∂T)
8   if IsBelowStorageLimit(glycogen) then
9      glucose ← ProduceGlucose(fat, ∂T)
10  if Ek > 0 then
11     glucose, Ek, O2  ← ProduceATP(aerobic, glucose, O2,  Ek, ∂T, eff)
12  if Ek > 0 then
13     adp, Ek  ← ProduceATP(anaerobic, adp, O2,  Ek, ∂T, eff)
14  if Ek > 0 then
15     glucose, Ek, 0,eff ← ProduceATP(anaerobic, glucose, O2,  Ek, ∂T, eff)
16  glycogen ← glycogen + glucose
17  glucose ← 0
18  return eStore
```

Table 3.2-3: Runner.ProduceKineticEnergy Algorithm

The following algorithm now presents how these different energy sources go about fulfilling the E_k request. Table 2.2-1: Muscles energy sources conversion) is useful in accounting for most of what goes on in here.

The algorithm produces ATP from either glucose or adp source depending on the eType requested. It does so anaerobically or aerobic depend on the rType requested. This process is inefficient, such that, it wastes energy through heat. This inefficiency can increase based on the lactic energy produced. See lines 19,28-29 for the handling of efficiency.

So in order to produce the energy required E_k, line 3 takes into account the inefficiencies. Thereon, line 15-22, atp is produced using the rType and eType requested,. These chemical conversions reduce the energy source and the oxygen (if done aerobically).

Lines 24-27, handles the energy that could not be fulfilled by the conversion. The functions

- ProduceAtpAerobicallyFromGlucose,
- ProduceAtpAnaerobicallyFromGlucose, and
- ProduceAtpAnaerobicallyFromADP

take into account the balancing of chemical equation and product rate described in section 2.2.2. The conversions also assume the Law of Conservation of Energy.

```
1   function ProduceEnergy(rType, eType, O2, Ek, ∂T, eff)
2    returns new energyStoreResidual of runner,
3             new  Ek Residual of runner,
4             new oxygenResidual of runner,
5             new efficiency
6    ínputs: O2, oxygen in blood available to muscles of runner
7            Ek, kinetic energy required by the muscles
8            ∂T, the time step
9            eff ,the efficiency of the conversion of glucose to atp
10           rtype, the type of conversion either aerobic or anaerobic
11           eType, the energy type
12
13  atpNeed ← Ek / efficiency
14
15  if IsAerobic(rType) and O2> 0 and IsGlucose(eType) then
16    atpProduced, oxygenResidual, energyStoreResidual ←
        ProduceAtpAerobicallyFromGlucose(atpNeed, eType, O2)
17
18  if IsAnaerobic(rType) and IsGlucose(eType) then
19    atpProduced, lacticAcidResidual, energyStoreResidual ←
        ProduceAtpAnaerobicallyFromGlucose(atpNeed, eType)
20
21  if IsAnaerobic(rType) and IsADP(eType) then
22    atpProduced, energyStoreResidual ←
```

```
        ProduceAtpAnaerobicallyFromADP(atpNeed, eType, O2)
23
24  if atpNeed > atpProduced then
25    Ek produced ← atpProduced × efficiency
26  Else
26    Ek produced ← Ek
27  Ek Residual ← Ek − Ek produced
28  if lacticAcidResidual <> 0
29    efficiency ← lacticAcidResidual / lacticAcidFactor
30
31  return energyStoreResidual, Ek Residual , oxygenResidual, efficiency
```

Table 3.2-4: Runner.ProduceEnergy Algorithm

3.2.4 Motion actuation

The previous discussions now allow us to describe the essential parts of motion actuation in concrete terms. The algorithm below puts together breathing, moving, and producing kinetic energy of a runner in a time step of motion. This completes the modeling of actuation of realistic motion.

```
1   function ActuateMotion(runner, acceleration, direction, ∂T)
2     returns  new velocity vector of the runner,
3              new position vector of the runner,
4              newEnergyStore of the runner
5
6   input: runner, competing in the athletic competition, contains his speed s, and
                 energyStore e
7          acceleration, magnitude of the acceleration
8          direction, direction of the acceleration
9          ∂T, the time step that elapsed since the previous actuation
10    local variable: O2, the amount of oxygen in the blood for the muscles
11  O2 ← Breathe(∂T, s[runner])
12  velocity, position, Ek ← Move(runner, acceleration, direction)
13  eStore← ProduceKineticEnergy(e[runner], O2, Ek, ∂T)
14  return velocity, position, eStore
```

Table 3.2-5: Runner.AcuateMotion Algorithm

3.3 Modeling sensation and perception for realistic motion

The environment dictates how motion can successfully be brought about. It is the state of the environment that dictates the motion decision-making process that yields an action. For runners, the environment consists of internal body functions and external entities. In particular it consists of the following:

- The Competition
- Ego
- Fatigue

In this section, we shall describe these elements in terms how runner senses and perceives them. It is important sensation and perception is modeled correctly. First, because it allows us to capture and characterize the correct percept needed for decision-making. Secondly, it allows us create agents that are fair, or rather are not perceived as cheating. Creating a sensation and perception model that allows non-player to access to a lot more information than a normal human being can comprehend, may lead to an unfair advantage. Here, we will describe the AI properties that are critical in building successful agent programs that can produce optimal results. For the sake of complexity, passive sensing is assumed.

3.3.1 The competition

"Our perceptual systems may have evolved to provide us with a sense of space that is not totally accurate but accurate enough to allow us to navigate the world"

(Matlin & Foley, 1997)

The competition in the athletic environment is perceived mainly in terms of the egocentric distance. Although not much understood on how distance is perceived, the eye can perceive the egocentric distance between an object of interest and the observer, using several visual cues (Matlin & Foley, 1997). The complexity of distance perception increases especially when the observer or the object is also moving. Hence perceiving the distance to the fixed finish line is easier than perceiving the distance to a moving competitor. With this insight it is therefore important we include this **partially observable** nature in non-player character agents so that they are not perceived as cheating.

In an athletic competition, more than one runner participates in a race. In the middle distance events up to 18 runners compete (Athletics). Each runner attempts to win the race. So success for one means failure for the other. AI describes such an environment as **multi-agent**.

This partially observable, multi-agent environment also results in an environment that is also **stochastic**. Suppose a runner A while overtaking decides to increase acceleration in order to take outright lead in the next time-step. However the other runner B also decides to increase acceleration at the same time in order to also take outright lead. This would cause none of the runners to take outright lead. Hence either runner cannot guarantee achieving a certain state based on the current state and action. This aspect introduces **uncertainty**.

Another characteristic of this competition is that it is **dynamic.** The environment is in continuous change. At every time step, the race is in continuous change. Hence each runner is required to act at every response time.

Sensing the competition information can either be done visually by observing what is the runners line of sight, or by detecting location of activity using hearing and turning to it, if runner are not in direct line of sight.

In real terms, there may be many runners, making it hard to sense all of them at every time step. So realistically we can assume that a runner can at least sense and perceive the most important competitor. That is naturally the leader. In case he himself is the leader, he may only be interested in keeping track of the runner right behind him.

The following algorithm describes perception of competitor can be implemented. The percept is a discretized form of the egocentric distance from the primary competitor.

```
1   function PerceiveCompetitors(runner, track, runners)
      returns percept of competitors
2   input: runner, athlete who is perceiving the competition
3          track, is the athletic track containing positional information and rules
4          runners, are the set of runners participating in the competition
5
6   if IsLeader(runner, runners) then
7     return PositionOfSecondRunner(runners)
8   return PositionOfLeader(runners)
```

Table 3.3-1: Runner.PerceiveCompetitor Algorithm

3.3.2 Ego

This is the awareness of one's self with respect to the external world. In motion three aspects are important

- The perception of self with respect to the objective and,
- The perception of one's own speed.
- Awareness of one's self to the rules

The discussion of distance sensation and perception was made in the previous section. It was stated that perception of distance was always to some extent inaccurate but less so to fixed goals, in this case. So we can assume the perception of distance is also inaccurate but less so than to dynamic competitors.

As for perception of movement of one's self, humans perceive movement using a measure of velocity. It has been demonstrated that humans are capable of a surprisingly high accurate perception of motion. It has also been established that there is a **velocity detection threshold.** This threshold varies depending on the cues available to aid movement perception (Matlin & Foley, 1997).

These findings thus also reveal a partially observable nature in the way we perceive speed.

The following algorithm now presents how the speed and the goal percept can be implemented. One way to PerceiveSpeed can be implemented, is by simply discretizing the magnitude of the velocity to integer accuracy. Similarly, the PerceiveDistanceToGoal percept can be implemented by simply discretizing the distance to the finish line to a range of accuracies.

```
1  function PerceiveEgo(runner, track, runners)
     returns speed percept, goal percept
2  input: runner, is the athlete who is perceiving the competition, and has
                velocity vector v, and position vector p.
3          track, is the athletic track containing positional information and rules
4          runners, is the set of runners participating in the competition
5
6  return PerceiveSpeed(v[runner]), PerceiveDistanceToGoal(track, p[runner])
```

Table 3.3-2: Runner.PerceiveEgo Algorithm

3.3.3 Fatigue

The last percept required for motion is fatigue. Fatigue represents the runners percept of the energy he has remaining available for motion. The brain can state the energy level of the body. Cues such as the carbon dioxide level, lactic acid, and breathing rate provides this information (Body Atlas, 2006). We can safely assume that it is of a partially observable nature.

The following algorithm presents the percept for fatigue. It simply perceives the energy available based on the three constraints.

- Glycogen level
- ADP level
- Efficiency

It is unnecessary to track fat, since one can always assume its presence. Whilst glucose is actually a function of glycogen and fat hence can is always derived from the two sources. Efficiency represents the amount of lactic acid levels in the blood which cause the body to become inefficient in producing the necessary kinetic energy.

Humans perceive the level of these attributes through feelings of fatigue; hence the percept is of a partially observable nature. One way to represent the percept is to introduce noise by discretizing the output, represent as a numerical value.

```
1   function PerceiveFatigue(runner)
2     returns adpPercept, glycogenPercept, efficiencyPercept
3     inputs: runner, the athlete who is sensing the environment, who has an
                  energyStore e containing adp, glycogen, and efficiency
4
5   return
6       PerceiveEnergy(adp[e]),
7       PercieveEnergy(glycogen[e]),
8       PerceiveEfficiency(efficiency[e])
```

Table 3.3-3: Runner.PerceiveEgo Algorithm

3.3.4 Implementing a sensation & perception

We can now combine together all the sense and percepts for runners and describe the sensation and perception algorithm below. The function returns a percept given the input from

the track and runners. Breaking the rules or blacking out result in disqualification percept. Crossing the finish line results in an evaluation of his degree of winning or losing. At all other intermediate states in the race, the runners' percept are a composite of ego, competition, and fatigue.

```
1   function Sense(runner, track, runners) returns percept of motion
2    input: runner, the athlete who is sensing the environment
3           runners, is the set of runners participating in the competition
4           track, is the athletic track containing positional information and rules
5    local variables: percept, the current motion percept
6
7   if EnergySourcesHaveBeenExhausted(runner)
8      percept ← DISQUALIFIED;
9   else if HasCrossedFinishedLine(runner, track)
10     if CannotStopWithoutBlackingOut(runner)
11        percept ←DISQUALIFIED
12     else if not WinnerIsClaimed(runners)
13        percept ← "WON" + runner.PerceiveCompetitor(runners)
14     Else
15        percept ← "LOST" + runner.PerceiveCompetitor (runners)
16  else if runner.HasBrokenRacingRules(track)
17     percept ← DISQUALIFIED
18  Else
19     percept ←
20        runner.PerceiveEgo(track) +
21        runner.PerceiveFatigue(runner.energyStore) +
22        runner.PerceiveCompetitor(runners)
23  return percept
```

Table 3.3-4: Runner.Sense Algorithm

The composites of ego, competition, and fatigue sensation can be broken down into the following attributes presented in the table below. The table also includes examples of how the percept values can be presented to match realistic perceptual characteristics such as partial observability.

Motion Senses	Description	Percept Values
Distance	Discretized to integer values assumed to be meters. Interval increase with distance. Maximum distance is the marathon distances.	1, 2, 3, 4, 5, 10, 20, 60, 100, 200, 300, 400, 600, 800, 1000, 1500, 3000, 5000, 10000, 50000
Speed	Discretized into integer values of meter per second.	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
Competitor	Discretized into integer values. Relative distance range to competitor presented in brackets.	FAR_AHEAD (10..), AHEAD (5..10), JUST_AHEAD (0..5), JOINT_LEADER (0), JUST_BEHIND (-5..0), BEHIND (-10..-5), FAR_BEHIND (-50..-10), VERY_FAR_BEHIND (-100..-50), EXTREMELY_BEHIND (-200..-100), IMPOSSIBLY_BEHIND (..-200)
ADP	Discretized in integer values of energy measured. Ten equidistant ranges presented.	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Glycogen	Discretized in integer values of energy measured. Ten equidistant ranges presented.	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Efficiency	Discertized into 5% ranges	0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75

Table 3.3-5: Runner motion percept values

We also observe that the percept is of a Markovian nature, specifically non-deterministic Markov Decision Process. The percepts returned, by the Runner.Sense are sufficient to predict what action to take, without having to inspect the entire percept history of the agent. Having to inspect the percept history would make the problem intractable especially where the game loop place a time constraint for time spent processing game logic.

3.4 Modeling acceleration decision making for realistic motion

We now move on to modeling decision-making for realistic motion. Motion relies on the brain to decide how much acceleration should be produced by the muscles to cause the body to move. Acceleration is a vector quantity consisting to two parts, the magnitude and, the direction. These two work together in combination to produce optimal results in motion. The choice of direction during motion is an area that has occupied game programmers for a long time now for obstacle avoidance, chasing and evasion strategies. Various methods have been developed including those that use rule-based fuzzy logic, potential function-based movement, path finding and way point techniques (Bourg & Glenn, 2004). In an athletic environment, direction is used for avoiding other runners, and to navigate the tracking using the shortest path. Simple techniques such as waypoints can be used to overcome the athletic problem. This is done simply by choosing the inner track if it is unoccupied. In this thesis, we will focus only on the choice of the magnitude of the acceleration and assume known techniques to handle the direction aspect separately.

In the previous sections, we described the game, the actuation, the environment, and sensation and perception for realistic motion. It was also demonstrated that the environment was dynamic, stochastic, sequential, partially observable, and multi-agent, which in AI terms is regarded the most complex environments (Russel & Norvig, 2003).

Agent Type	**Performance Measure**	**Environment**	**Actuators**	**Sensors**
Runner	Consistency in winning an athletic competition	Competitors and self in motion Skeletal muscles and their chemical energy stores Breathing, heartbeat and blood for oxygen circulation The running track and competition rules	Acceleration / Decelerate using Skeletal Muscles Turn using Skeletal Muscles	Distance to finish line Energy remaining Distance to primary competitor Current speed

Table 3.4-1: PEAS for an athletic competition

At this stage we can establish the performance measures, environment, actuators and sensor (also known as PEAS) for an agent in an athletic competition. These PEAS will play an important role in helping us overcome basic design issues when modeling for a rational agent. Table 3.4-1: PEAS for an athletic competition above summarizes them.

This section we will address how we deal with the decision making process that produces a magnitude of acceleration at each time step so as to accomplish the performance measure that is defined. We will begin discussing how knowledge can be acquired that achieves the performance measure. Two methods in AI, Q-learning and neural networks will be the primary focus in this section, and hence we eventual propose a solution that consists of a combined model of the two methods.

3.4.1 Learning in the absence of prior domain knowledge

The athletic competition requires a runner to perform a sequence of actions in the competition. At each point during the race, the runner perceives the environment and almost instantaneously is required to instruct the skeletal limbs muscles to accelerate in order produce a force that results in motion. The choice of acceleration at each point in the race plays a fundamental part in the overall outcome. At this point we do not have the domain knowledge to produce a sequence of actions that lead to achieving the performance measure of winning in a consistent manner. Naive methods would require us to maintain a state history of each episode and measure each one against the other. It would require huge resources in terms of time and space complexity to handle a multi-agent and dynamic environment such as this.

Reinforcement learning allows us to model an agent program that can learn unsupervised, that is, without any prior domain knowledge. Reinforcement methods are built upon dynamic programming methods. Dynamic programming, are powerful programming techniques for solving optimization problems by recursively building solution to larger sub-problems (Kleinberg & Tardos, 2006), thus have low computational demands in moderate state-spaces. Reinforcement method, however poses the following additional requirements if they are to work.

1. the environment must provide feedback in terms of a reward
2. the percepts must possess Markov property

The requirements combined are what known as Markov Decision Process. Fortunately, we already concluded in section 3.3.4 that the runners' percept of the environment also possesses the non-deterministic Markov Decision Process. What remains is the reward which we discuss in the next section.

3.4.2 Defining rewards percept for an athletic competition

We now introduce the reward percept into the athletic agent. A reward is a signal provided by the environment to the agent to indicate the value of the action the agent has taken in a state. Rewards carry a numerical value; higher values encourage agents to aspire to reaching them whilst lower ones do otherwise. It is easy relate the reward percept to an athletic competition, especially at the end states of a race. High rewards would indicate winning, whilst lower one would indicate losing. An earlier discussion in section 2.3.1 provided a detailed discussion on the role played by the reward in Q-learning.

The following table shows an example on how rewards can be perceived from the environment in an athletic competition. Winning is reward positive value and losing otherwise. Not completing the race, by either blacking out or breaking the competition rules is least rewarded. Losing or winning can also be present as a degree by the reward percept. In the athletic environment winning reward more if the nearest is even further away. Losing by large distance is also poorly reward as opposed to losing by a small margin of distance. This is done so are to encourage winning in a convincing fashion.

State	**Reward**
Disqualified	-1.0
Losing far behind winner	-0.5
Losing just behind winner	-0.1
Winning just behind nearest winner	0.5
Winning far ahead nearest loser	1.0
All other states in the race	0.0

Table 3.4-2 Examples of rewards for runner

To implement this reward percept, one can simply extend the Table 3.3-4: Runner.Sense Algorithm to include one an extra return value containing the percept of the reward. The table below shows the extended version containing the handling of the reward percept.

```
1   function Sense(runner, track, runners) returns percept of state, percept of reward
2    input: runner, the athlete who is sensing the environment
3           runners, is the set of runners participating in the competition
4           track, is the athletic track containing positional information and rules
5    local variables: rewardRercept, the reward percept from the environment
6                     statePercept, the runners motion percept
7                     compStatePercept, the percept of the competitors
8                       compRewardPercept, the percept of reward to the competition
9
10  if EnergySourcesHaveBeenExhausted(eStore) then
11    statePercept ← DISQUALIFIED;
12    rewardPercept ← -1;
13  else if HasCrossedFinishedLine(runner, track) then
14    compStatePercept, compRewardPercept ← PerceiveCompetitor(runner,runners)
15    if CannotStopWithoutBlackingOut(runner) then
16      statePercept ←DISQUALIFIED
17      rewardPercept ← -1;
18    else if not WinnerIsClaimed(runners) then
19      statePercept ← "WON" + compStatePercept
20      rewardPercept ←compRewardPercept
21    else
22      statePercept ← "LOST" + compStatePercept
23      rewardPercept ←compRewardPercept
24  else if HasBrokenRacingRules(runner, track) then
25    statePercept ← DISQUALIFIED
26    rewardPercept ←-1
27  else
28    statePercept ←
29      PerceiveEgo(runner, track) +
30      PerceiveFatigue(runner) +
31      PerceiveCompetitor(runner, runners) +
32    rewardPercept ← 0
33  return statePercept, rewardPercept
```

Table 3.4-3: Runner.Sense Algorithm

Lines 5, 6, 7, 9, 12, 14, 17, 20, 23, 26, 32 and 33 have been added or altered to include the handling of the reward. Line 14 receives a reward in the form of a variable when the runner has finished the race. Table 3.4-2 Examples of

rewards for runner) is used to determine the reward which depends on the degree of separation

3.4.3 Acquiring prior domain knowledge

At any real world athletic event, athletes are expected to be competent about the competition prior to its commencement. In A.I. it is broadly termed as prior domain knowledge. In this section we will describe how we can achieve modeling of this behavior.

Prior knowledge is most commonly acquired through prior experience. One method used to acquire this knowledge is through training. We can allow the agent program to simulate an actual event and participate in it. However a normal race is multi-agent in nature. To solve this, we could include several copies of the running agent, in the same race, acting independently but sharing the same repository of domain knowledge.

In section 2.3.1, we introduced Q-learning, a reinforcement learning method capable of learning optimal behavior in environment in which it has no prior knowledge. To do this, Q-learning maintains a value called a Q-value for every state-action pair.

This Q-value is updated using Equation 2.3-1 Q-value update for a non-deterministic MDP. The choice of this update method suits the stochastic nature of this athletic environment (Mitchell, 1997). The Q-value updated in this actually represents a probabilistic value of choice for an action, which can be interpreted as the probability of that action achieving the optimal solution. The algorithm below describes how the Q-learning update can be implemented. It also computes the square of the difference of the error between the old and the new value. The error can be used to measure the rate of convergence during training for prior knowledge acquisition.

1 **function** Qlearn(*currState, prevStateAction*) **returns** new qValue, the square of the error
2 **inputs:** *currState*, which contains a set of action **a**, and reward **R**
prevStateAction, which contains the Q-value **qValue**
3 **local variables:**
4 *error*, stores the square of the difference between the old and new qvalue
5 *newQValue*: stores the new Qvalue
6 $newQValue \leftarrow qValue + \alpha\,(R + \gamma\, max_a qValue - qValue)$
7 $error \leftarrow newQValue - qValue$
8 **return** $newQValue, error^2$

Table 3.4-4 AgentState.Qlearn Algorithm

But in order to learn, it needs to find a way to visit as many states and perform their actions so as to discover which action yield good long-term rewards. Discovering the value of state-actions is known as exploration.

One of the most effective exploration methods used, is the softmax action selection methods, especially the Boltzmann Distribution, in the form,

$$\frac{e^{Q_t(action)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

Equation 3.4-1 Boltzmann Distribution for Softmax Action Selection

Where,

$Q_t(action)$ is the Q-value of the $action$ in state t.

τ is the temperature.

In this probabilistic method, when an agents (runner) is provided with a state percept and list of actions (accelerations) to choose from, he chooses the acceleration whose probability of achieving success is less than the number of times it has been selected against the other possible actions in that state. This guarantees all actions will be explored enough times corresponding to their success rates. This method however requires that the frequency of state-action pair visits be tracked. Fortunately only when no prior knowledge is available is such exhaustive exploration needed. Alternative weaker methods exist which do not require frequency table to be maintained. See (Sutton S & Barto G). These become useful once domain knowledge has been acquired.

The temperature τ in Equation 3.4-1 Boltzmann Distribution for Softmax Action Selectionis used to control the balance between exploitation and exploration. During training for prior knowledge acquisition a higher value would be set to encourage exploration whilst on the actual competition τ would be set to a low value to encourage exploitation.

The algorithm below, implements the description given above. Note that, whenever an action is chosen, the frequency f is updated as well (line 14).

```
1   function DecideAcceleration(state) returns an action
2    inputs: state, which contain a set of action actions each containing a qValue Q
3                    and frequency time it has been selected f
4    local variables: selectedAction, store the action that has been selected.
5    statics: τ is the temperature
6
7   selectedAction ←null
8   foreach a in actions
9     if selectedAction = null then
10      selAction ← a
11      Break
12    if f[a] = 0 then
13      selectedAction ← a
14      break
15    if e^{Q[a]/τ} / Σ_{b=1}^{n} e^{Q[actions (b)]/τ} < f[a] / Σ_{b=1}^{n} f[actions (b)] then
16      selectedAction ←action
17
18  f[selectedAction] ← f[selectedAction] + 1
19  return selection
```

Table 3.4-5 AgentState.BoltzmanAction Algorithm

So at this point we now know how make an agent explores and learns, what follows, is the when to learn, where to capture the domain knowledge and, how to represent that knowledge of successfully winning the race. There are two ways to do this.

- Online, meaning adjusting the Q-learning lookup table during the race
- Offline, meaning capturing the percept history during the race, and then adjusting the Q-learning lookup tables, after the race (episode) is complete.

Offline learning is favored, in the case of capturing prior knowledge because it has a faster convergence rate than does online. This is because of its rippling nature from the actual goal, as opposed to online learning which can end up updating only one Q-value in an episode even though several states were visited.

The algorithm below shows how Q-learning can be updated to serve as repository of domain knowledge. It also updates the cumulative square of the error in line 16 and keeps track of the total number of state-actions performed. This information can be used to determine whether

the algorithm actually convergence, by computing the root mean square. Convergence demonstrates that the algorithm will eventually arrive to the optimal solution.

```
1   function OfflineQLearnEpisodePerceptHistory(
      runner, qTable, episodePerceptHistory)
2     returns updated Q-table, sum of the square of error
3     inputs: runner, the athlete who experienced the percept, contains frequency f of
4                   states visited, and sum of square of the Q value error e
5           qTable, is the Q-learning look table containing states s. Each of the
6                   states contain a set of actions with their corresponding Q-values.
7           episodePerceptHistory,is a stack of states-action pair representing the
8                   percept history of the episode
9     local variables: currState, is the current state s and action a pair
10      prevState, is the previous state s' and action a' pair
11
12  currSA, prevSA ← null
13  f ← f + Count(episodePerceptHistory)
15  while prevSA in Pop(episodePerceptHistory) do
15    if currSA is not null then
16      qTable[s][a], newSqOfError ← QLearn(qTable[s][a], qTable[s'])
17      e ← e + newSqOfError
18    currSA ← prevSA
19  return qTable, e
```

Table 3.4-6 Runner.OfflineQLearnEpisodePerceptHistory Algorithm

And below we show how the AthleticGamesEpisode is adjusted to cater for capturing the episode percept history and final offline Q-learning. In the algorithm below, the *qTable* variable is the domain knowledge represented as a Q-lookup table. Line 22 performs the learning part and agentStates Q-values are then updated.

Therefore to capture prior domain knowledge using the structure, we simply set a high temperature, repeat the AthleticGameEpisode algorithm until a certain threshold of performance is achieved by the runner. In the actual training of the athletic competition high exploration and high exploitation were alternately applied, the temperature set were 2.0 and 0.01 respectively. In the next section we shall introduce how we measure performance.

```
1   function AthleticsGameEpisode(gameSettings)
2    inputs: gameSettings
3    local variables: track, contains the athletic environment and its competition
                        rules
4                    runners, the set of runner competing in the race each with a
                        velocity vector v, position vector p, and energy store e,
                        and race status rs
5                    ∂T, the time step that elapsed since the previous game loop
                    episodePerceptHistory, contains percept history of the episode

6   InitialiseAthleticEpisode
7      track ← InitializeRunningTrack(gameSettings)
8      runners ← AddRunnersRandomlyOnToTrack(gameSettings, track)
9      episodePerceptHistory ← {}
10  AthleticEpisode
11     Repeat
12        foreach r in runners
13           if rs[r]is InRace or NotStarted then
14              statePercept, rewardPercept ← Sense(r, track, runners)
15              if statePercept not in qTable then
16                 Add(qTable, statePercept, rewardPercept)
17              acceleration,direction←DecideAction(r, percept, qTable)
18              v[r],v[p],r[e] ← ActuateMotion(r, acceleration, direction, ∂T)
19              episodePerceptHistory.Push(statePercept, action)
20     until RunningEpisodeStoppingCriteria(runners, track)
21  FinalizeAthleticEpisode
22     qTable ←
          OfflineQLearnEpisodePerceptHistory(qTable, episodePerceptHistory)
23     PresentResults(runners)
```

Table 3.4-7: The Athletics Competition Algorithm

3.4.4 Implementing the critic for performance measure

We know have an agent who can acquire prior knowledge which can potentially converge toward the optimal solution with more exploration. In stochastic environment however, convergence to the optimal solution is not guaranteed. And since we do not have an agent that behaves optimally using any other method, it becomes hard to judge when an agent has acquired enough to be behaving rationally. In the following discussion we shall propose how

to introduce a critic, upon which the agent program can use as a benchmark of measuring success in its learning experiences.

In the introduction section, we mentioned that most of the games programs today use rule-based and fuzzy logic to implement the decision-making process for motion. We can use current forms of expressing decision making as a benchmark for the success criteria for this learning agent program.

A pace setter running strategy can be one example of fuzzy logic agent program that can be implemented. Pace setters in athletics, are runners who use the strategy of sprinting off in the beginning of the race in order to take early lead, and then gradually slowing down as energy resources deplete. This strategy is simple because, it bases all decision making only on the energy and speed percept of the runner, and requires no prior knowledge. It would get nastier with more percept information added. See section 3.3, for a discussion on the percepts used for motion. The following algorithm presents a fuzzy rule-based implementation of such a strategy. The algorithm returns the magnitude of acceleration discretized into values [-2, -1, 0, 1, 2]. Basically the agent will accelerate to achieve high speed when it has a high energy level, and decelerate to lower speed as energy depletes.

```
1   function DecidePaceSettorAcceleration(percept)
2     returns magnitude of the acceleration
3     inputs: percept, consists of the runners own speed and energy
4
5   if  energy is ABUNDANTENERGY and speed has not reached PEAK then
6       return 2
7   if energy is MODERATEENERGY and speed is HIGHSPEED then
8       return -1
9   if energy is MODERATEENERGY and speed is LOWSPEED then
10      return 1
11  if energy is LOWENERGY and speed is HIGHSPEED then
12      return -2
13  if energy is LOWENERGY and speed is LOWSPEED then
14      return -1
15  if speed is WALKINGSPEED then
16      return 1
17  return 0
```

Table 3.4-8: The DecidePaceSettorAcceleration Algorithm

Now once we have the pacesetter program we can use it to implement the critic program which measures success of prior knowledge acquisition. One way to implement the critic is to allow it to simulate several game episodes whereby several of the runners use the Table 3.4-8: The DecidePaceSettorAcceleration Algorithm) strategy whilst one runner uses the Table 3.4-5 AgentState.BoltzmanAction Algorithm). During the run of the critic program, the temperature of the boltzman action algorithm is set to allow for maximum exploitation.

The numbers of wins made by the Q-learning agent are then tracked. The algorithm then returns a value stating whether winning ratio has crossed the winning benchmark ratio. The algorithm below implements the idea of such a critic using the description above. This algorithm can also be extended to track the rate of convergence using the root mean square.

```
1   function CrossValidate(gameSettings)
2     inputs: gameSettings
3   for (episode ← 1..numOfTrainingEpisodes) then
4     AthleticGameEpisode
5      if RunnerUsingStrategyHasWon(QLearn) then
6         success ← success + 1
7     else
8         failure ← failure + 1
9   winningRatio ← success / (success + failure)
10  return winningRatio > WINNINGBENCHMARKRATIO
```

Table 3.4-9: The CrossValidate Algorithm

Now we have all the components available to implement the complete program that can perform the domain knowledge acquisition. What is needed to do is, perform a sufficient large number of game episodes of using several runners each using the exploration technique for action selection using Boltzmann algorithm from Table 3.4-5 AgentState.BoltzmanAction Algorithm, which at the end of each episode performs Q-learning. It is important to gradually reduce the temperature for successful learning. Once that is done, the critic implemented above is invoked. If successful, the agent has met the level of knowledge enough to meet expected future success.

```
1   function AthleticPriorKnowledge(gameSettings)
2     returns agentState table that contains the updated Q-values
3
3   repeat
4       for (episode ← 1..numOfEpisodes)
5           AthleticGameEpisode(gamesSettings)
6       if  CrossValidate(gamesSettings) then
7           return agentStates
8       numberOfFailures ← numberOfFailures + 1
9       if numberOfFailures exceeds a certain threshold
10          return null
11  until true
```

Table 3.4-10: The AthlecticPriorKnowledge Algorithm

3.4.5 Using experience to predict behavior for a larger state space

Q-learning was used in the previous section to capture prior knowledge on how to behave optimally in this complex athletic environment where we had no domain theory. This domain knowledge was captured by means of a look-up table consisting of a state-action pair which had an associated Q-value to it. In an athletic running environment, however maintaining the domain knowledge quickly becomes large and hence intractable as the distance of running is increased, causing the state space to explode. This can also have an adverse effect on the time complexity for decision making this dynamic environment. As an example using the percepts described in Table 3.3-5: Runner motion percept values) can explode to 30 million state-action pair if an exhaustive state search is performed. It would also take much longer time for convergence to take place during training. The table below illustrates the growth on the size a function of the distance to run a standard athletic competition.

Distance to run	**States** (distance,adp,atp,efficiency,speed,competitor)	**State-action** (acceleration=-2,-1,0,1,2)
100	$9 \times 10 \times 10 \times 15 \times 20 \times 8 = 2160000$	$2160000 \times 5 = 10800000$
200	$10 \times 10 \times 10 \times 15 \times 20 \times 9 = 2700000$	$2700000 \times 5 = 13500000$
400	$12 \times 10 \times 10 \times 15 \times 20 \times 10 = 3600000$	$36000000 \times 5 = 18000000$
800	$14 \times 10 \times 10 \times 15 \times 20 \times 10 = 4200000$	$42000000 \times 5 = 21000000$
1500	$16 \times 10 \times 10 \times 15 \times 20 \times 10 = 4800000$	$48000000 \times 5 = 24000000$
5000	$18 \times 10 \times 10 \times 15 \times 20 \times 10 = 5400000$	$54000000 \times 5 = 27000000$
10000	$19 \times 10 \times 10 \times 15 \times 20 \times 10 = 5700000$	$5700000 \times 5 = 28500000$
42000	$20 \times 10 \times 10 \times 15 \times 20 \times 10 = 6000000$	$6000000 \times 5 = 30000000$

Table 3.4-11: Space complexity for q-value lookup table as a function of distance to run

Now if every individual runner would store his own motion domain knowledge using a look-up table, the games loop would be incapable to meeting the time requirements, due to the time-and-space complexity of the q-tables alone.

In this section, we shall see how we can use neural networks to model this domain knowledge such that it solves the problem described above whilst retaining the power of Q-learning on a sequential, stochastic, partially observable, multi-agent, and dynamic environment.

3.4.6 Using neural networks to represent Q-learning

As was introduced in section 2.3.2, neural networks can used to classify statistical probabilistic patterns in a compact forms that also allows for generalization given the right training. This is exactly what our problem is about; can we classify actions in a probabilistic fashion?

But first we need to define a mapping from Q-learning to neural networks i.e. transform the state percept, Q-values, and Q-learning to an equivalent representation using a neural network.

1. *Mapping the q-table state percept to a neural network representation*

The input layer of a neural network is the equivalent of the Q-learning state percept. However, whereas the state percept of a Q-table is represented as a concatenation of all state percept attributes, the input percept of a neural network consists of decomposed percepts with each state attribute represented as an input node each taking as input, a numerical value.

In the case of motion, using Table 3.3-5: Runner motion percept values), the q-table state would be represented as

$$state\ percept\ =\ [Distance\_Speed\_Competitor\_ADP\_Glygcogen\_Efficientcy]$$

The individual percepts in the state percepts would then be transformed neural network input percepts in the following way.

- Distance percept was discretized into irregular intervals. Hence it is appropriate to create a node for each value of the distance percept. Specifically, a node for percept value for 1, 2, 3, 4, 5, 10, 20, 60, 100, 200, 300, 400, 600, 800, 1000, 1500, 3000, 5000, 10000, 50000. This results in 20 input nodes for the distance percept. Each of these nodes would accept either input the value 1 or 0.
- The speed percept was discretized in regular intervals, therefore is sufficient to use one input node that accept the discretized input of the speed.
- The competitor percept was discretized into 10 irregular intervals. Similar to the distance percept, the appropriate thing to do is create 10 input nodes for each percept value for FAR_AHEAD, AHEAD , JUST_AHEAD, JOINT_LEADER, JUST_BEHIND, BEHIND, FAR_BEHIND, VERY_FAR_BEHIND, EXTREMELY_BEHIND , IMPOSSIBLY_BEHIND. Each of these nodes would either accept the value 1 or 0.
- The adp, glycogen, and efficiency percept were all discretized to regular intervals, therefore similar treatment as with the speed percept would follow. One input node for each.

Pre-processing can improve the performance of a neural network (Bishop, 2007). We have already done some of that by the treatment made on the distance and competitor percepts. Another form of pre-processing that helps the network is to normalize all input values, so that all input node accept value having the similar order of magnitude. Linear transformation, i.e. scaling and translating, can be performed to achieve this. In our case, the following can be performed.

Percept	**Pre-processing transformation**
Distance	As discussed above, increasing nodes, that accepts only 1 or 0.
Speed	Can be scaled by dividing using the max speed humanly possible. For example, in our implementation, we would divide by 20.
Competitor	As discussed above, increasing nodes, that accept only 1 or 0.
Adp	Can be scaled by dividing using the max speed humanly possible. For example, in our implementation, we would divide by 10.
Atp	Can be scaled by dividing using the max atp humanly possible. For example, in our implementation, we would divide by 10.
Efficiency	Which is expressed as a percentage can be scaled such that it value lie between 0 and 1. For example, dividing a percentage by 100 achieves this.

Table 3.4-12: Pre-processing motion percept input for neural networks

2. *Mapping q-Values to a neural network representation*

The Q-values in a Q-table can be represented as values that are output from a neural network follow activation of the outer layer nodes. Since state contain a set of actions with associated Q-values, transforming this to a single neural network would therefore requires to be as many output nodes as there are actions. For instance in the case of the runner, the acceleration choice action at any state are [-2, -1, 0, 1, 2], hence five nodes would be constructed on the output layer. The nodes in the outer layer would therefore output Q-values when activated.

Whereas a mapping of state-action to Q-values was done through a look-up table, the equivalent is now achieved using a neural network by accepting pre-processed state

values through the input layer node, and perform activating the entire network to obtain corresponding Q-values as output.

The following diagram, illustrates the setting on the network based on the discussion above.

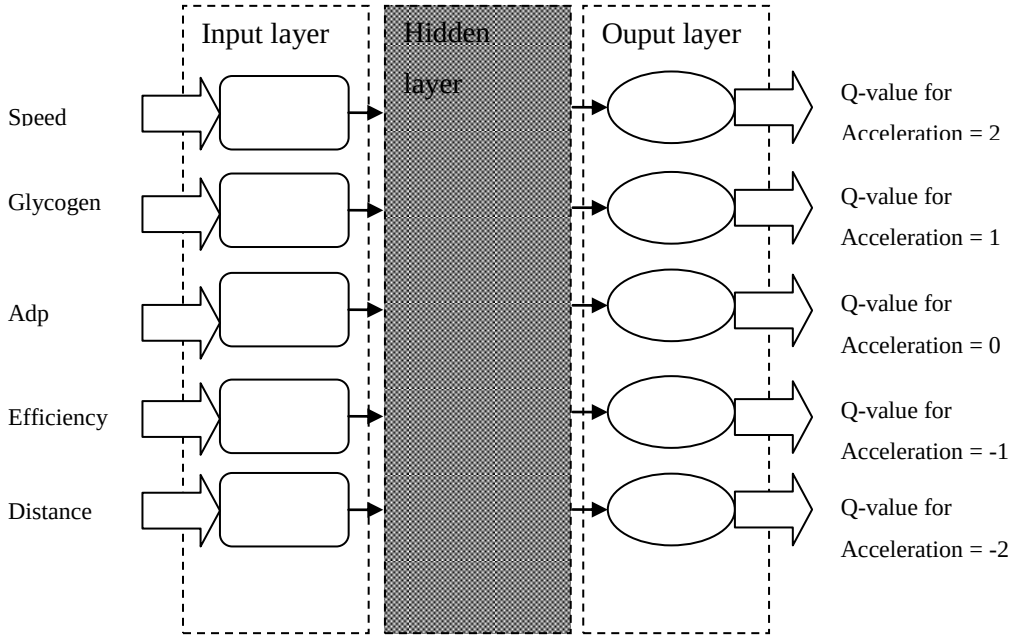


Table 3.4-13 2-layer neural network representing a Q-table

Note that based on the above discussion, the arrow flowing into the hidden layer, perform pre-processing on the input signal of the five percepts. The pre-processing may include splitting of the input signal into several nodes, converting into discrete values, normalization etc. For example, the distance signal is accepted through 10 input nodes each accepting 1 or 0.

The hidden layer is kept gray because the value of the weights flowing in and out of the layer, and the number of nodes is still unknown.

The outer layer is represented by 5 nodes each outputting a Q-value when activated to reflect the desirability of the acceleration action.

The activation is achieved using a roll-up operation. The algorithm below describes how this can be implemented. The Activ function in line 8, is from the Equation 2.3-2 Neuron activation function.

```
1   function NeuralNetworkRollup(neuralNetwork, statePercept)
2     returns stateOutput contain Q-value of the output layer
3     inputs:  statePercept, contains the input percept
4                neuralNetwork, contain L layers, and hence L is the outer layer
5
6   stateOutput ← {}
7   foreach node_i in NeuralNetwork(L)
8     stateOutput[i] ← Activ(statePercept, L, i)
9   return stateOutput
```

Table 3.4-14 NeuralNetwork.Rollup Algorithm

3. *Mapping q-learning to a neural network*

We just saw how the physical representation of the look-up Q-value table could be expressed as neural network mapping state to action through activation using the rollup algorithm. Such that, they earlier Q-Learning algorithm that assumed Q-values were stored in an explicit has to changed to retrieve Q-value by computing rollup.

The algorithm below shows how this can be implemented. Lines 6-7 now use Rollup to compute Q-values used to update the Q-value for the prevStateAction.

```
1   function QLearn(neuralNetwork, currState, prevStateAction_i)
      returns the new Q-value, the square of the error
2     inputs: currState, which contains a set of action a, and reward R
3             prevStateAction, which contains the Q-value qValue
3     local variables: qValues: stores Q-value of the prevState
                       newQValue: stores the new QValue
4                      error, stores the square of the difference between the old and new
                             Q-value
5
6   qValues ← Rollup(neuralNetwork, prevState)[i]
7   currStateQVals ← Rollup(neuralNetwork, currState)
8   newQValue ← qValues[i] + α (R + γ max_a currStateQVals – qValue[i])
9   error ← newQValue − qValues[i]
```

10	**return** $newQValue, error^2$

Table 3.4-15 AgentState.QLearn Algorithm

We cannot store the new Q-value that has been computed because we got rid of the Q-table. However what can now do is to update the neural network such that it can represent this mapping of state to Q-value. However to train a neural network using back-propagation neural network it needs to have an example consisting of an input x signal and a target value t. In our case, the input signal is the state and the target value is the new Q-value that has just been learnt using Q-learning. We can therefore change Backpropagation algorithm in Equation 2.3-5 Backpropation error in neural network weight updateto become,

$$\Delta(x,l,i) = \begin{cases} g'\big(activ(x,l,i)\big) \times (QLearn() - activ(x,l,i)) & l \text{ is an outer layer} \\ g'\big(activ(x,l,i)\big)\sum_{j=0}^{n} \Delta(x,l+1,j)w([l,i],[l+1,j]) & l \text{ is a hidden layer} \end{cases}$$

Equation 3.4-2 Backpropation error in neural network weight update

Where $t[i]$ is now replaced by $QLearn()$ and the rest stays the same as explained earlier.

A similar treatment has to be performed to the Boltzmann Algorithm in Table 3.4-5 AgentState.BoltzmanAction Algorithm exploration-exploitation. The Q-table references have to replace by the neural network and explicit look-ups to Q-values replace with the Roll-up function instead.

3.4.7 Achieving generalization using a neural network

Achieving the correct architecture for a neural network that generalizes well to its examples is an art in it own right. There is numerous literature offers a vast variety method on how to do so. There is a balance to be struck between generalization and accuracy. Initially we, assumed a specific class of architecture of neural network that we would eventually use. That is, the feed-forward fully connected multilayer neural network.

In fact we now narrow it to a specific type in that class, the 2 layer neural network, since studies show that (Bishop C. M., 2007) these can represent effectively any non-linear function given sufficient nodes in the hidden layer.

Hence the problem reduces to finding sufficient number of nodes in the hidden layer such that it represents pretty accurately the behavior that the Q-table could express, whilst generalizing to spaces that were not covered by the Q-table. Overdoing can lead to poor generalization since the network just memorizes due to the *curse of dimensionality*.

Growing algorithms can be used to determine the number of nodes is. These particular types of algorithms start off by training a network with a few nodes in the hidden layer and gradually increase the node until stopping criteria is reached. For our particular problem, a stopping criterion can be when the agent reaches a particular winning ratio threshold against the pace-setter program. We can therefore adjust the Prior Knowledge Algorithm Table 3.4-10: The AthlecticPriorKnowledge Algorithm such that it now handles neural networks.

Another problem that neural network encounter during training is that can end up converging into local minima which can result in poor performance. Initialization of weights and the learning rate become important factor in the network converging to good minima. One method to overcome the problem is to retrain using different value for initialization and setting the learning rate to a low value such that training does not escaping a good minima for more on initialization and minima see (Bishop C. M., 2007).

The algorithm below shows how this can be implemented. It starts off with a network with 1 node in the hidden layer. It trains the network and tests using different initializations of the weight. If several attempts all fail, it adds one more node in the hidden layer and performs the same thing again until a successful candidate is found, or the hidden layer becomes worryingly large.

```
1   function AthleticPriorKnowledge()
2     returns multilayer neural network that contains the updated Q-values
3     local variables: n, is the number of node in the hidden layer
4                      f, is the number of failed trials for  using n nodes
5                      i,j are the number of input and output layers
6   n ← 1
7   multiLayerNeuralNetwork ←InitializeNetwork(i, n, j)
8   repeat
9     for (episode ← 1..numberOfTrainingEpisodes)
10      AthleticGameEpisode(gameSettings)
```

```
11      if  CrossValidate(gameSettings) then
12       return multiLayerNeuralNetwork
13      f ← f + 1
14      if f exceeds a certain threshold then
15          if n exceeds a certain threshold then
16              return null
17          n ← n + 1
18          multiLayerNeuralNetwork  ←InitializeNetwork(i, n, j)
19          f ← 1
20  until true
```

Table 3.4-16: The AthlecticPriorKnowledge Algorithm

Chapter 4. Empirical results

In this section, we will present the results obtained from experiments made using an implementation the athletics running competition. We will begin by briefly describing the nature of the implemented competition. Following that we will present results based on the three type of running strategies that were implemented .i.e. the fuzzy rule-based agent, the Q-learning agent, and the combined Q-learning and neural network agent.

4.1 Overview of the actual athletic implementation used for the thesis

The implementation of the athletic game consisted of the following elements.

- The Running Track

  The track that was implemented and shaped exactly like the one found in Olympic athletic events. Its total running length was 400 meters (100m on the both straights and 100m on both half circled curves). It consisted of up to 10 concentric tracks, each measuring 1m wide. Athletes in the simulation were positioned in the track and allowed to run on within it, just as in the case of a real life Olympic event. Way-point logic was implemented to help the runner navigate the track. The environment was assumed to have a force of gravity = $9.8m/s^2$, a constant coefficient of friction and drag constant. The later are introduced to avoid entering a state of inertia. Hence they influence the energy usage even when running is at a constant acceleration.

- The Runner

  The athletes were implemented as points. The following value were assigned each to them at the beginning of the race.

Resource	**Initial value**	**Comments**
Mass	60 kg	Same for all runners. Affect force and energy calculations
Fat energy	100000 Joules	The amount assume abundant energy source that cannot be exhausted Its conversion rate could provide enough glycogen to sustain motion at 3m/s
Glycogen energy	3000..5000 Joules	Glycogen is replenished by excess fat that has been

		converted to glucose.
Adp	5000..10000 Joules	Magic number chosen that can sustain motion at high speeds for at most 100 meters. This source is not replenished.
Reaction time	0.22 seconds	Time between sensing and acting
Efficiency	Initially 75%	Affected by anaerobic use of glycogen
Rate of breathing oxygen	177777	Magic number chosen to allow aerobic respiration to sustain motion at speeds less the 4 m/s

Table 4.1-1: Runner settings in the actual implemenation

- The Competition

  The competition consists of a running event. The goal is run from a starting line on the track and attempt to win the race by finishing first. The distance to run is decided at the start of the race. A running event ends when all runners have completed the race by disqualification or crossing the finish line. The following rules are observed.

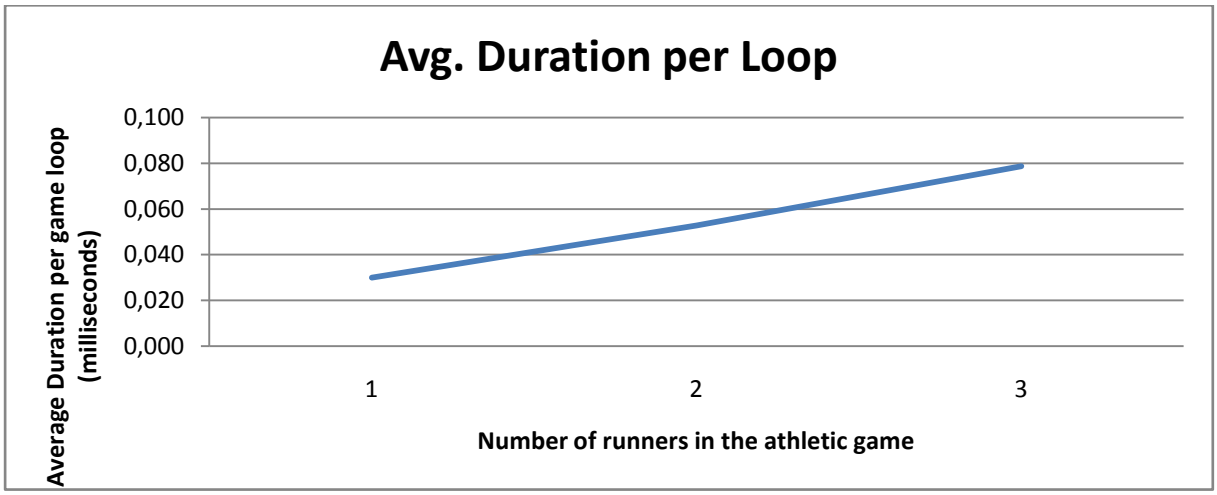
  a. Running at less than $2m/s^2$ results in a disqualification.
  b. Running in the opposite direction lead to disqualification.
  c. Overtake is allowed only from the outer side of the outer track
  d. Blacking-out during or after the race when coming to a stop lead to a disqualification.
  e. Up to 10 runners are allowed in same event. For most cases 3 runners were at most used.

4.2 Fuzzy Logic Rule Based Agent

We shall now present result obtained from an agent implemented using the pace settor algorithm Table 3.4-8: The DecidePaceSettorAcceleration Algorithm. This will reveal the behavior we expect the energy support.

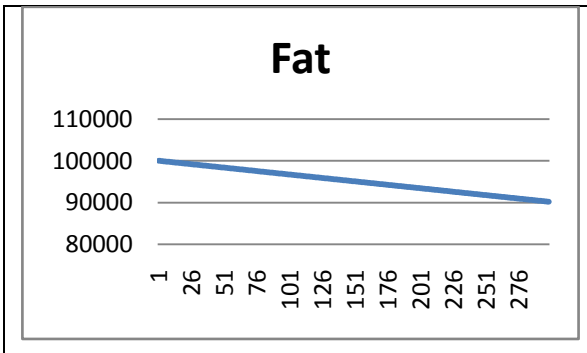
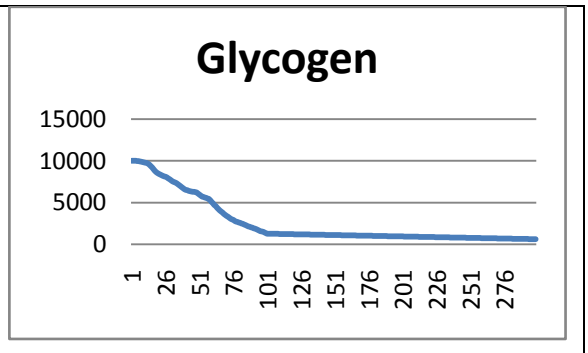
4.2.1 Fuzzy Logic computation time in Game Loop

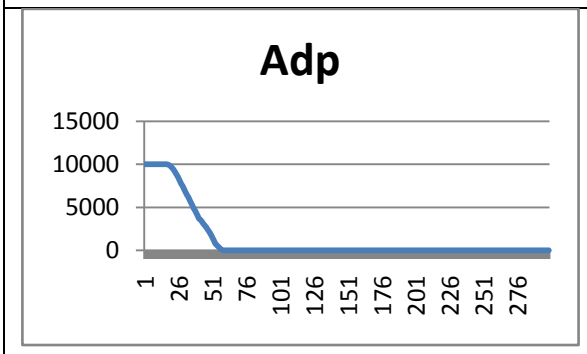
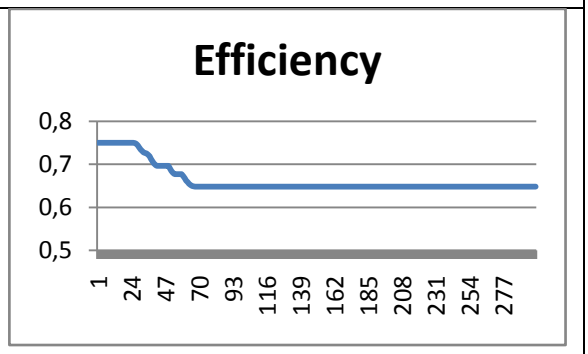
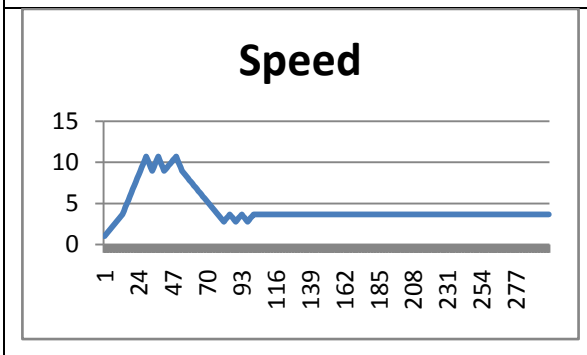
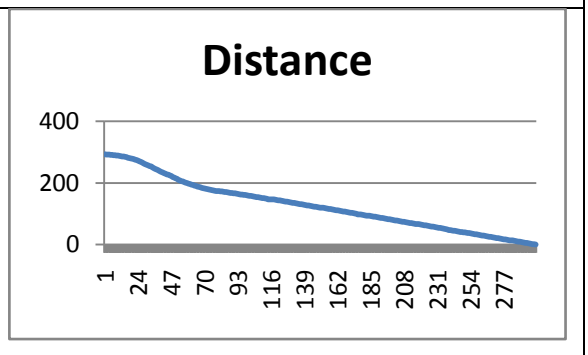
The graph below shows the average duration per game loop spent for varying number of runner in an athletic game episode. The threshold per game is 1000 milliseconds / 24 frames (game loops) = 41.67 millisecond per game loop. The growth is linear and it would require at least 1500 runner to reach the unacceptable limit.



4.2.2 Motion and its effect on key attribute

The following table gives an overview of the dynamic of motion attributes. It present motion performed by the pace-setter athlete agent running a 300m stretch. The logic is implemented using fuzzy rule-based logic described in Table 3.4-8: The DecidePaceSettorAcceleration Algorithm.

	
Fats falls constantly but very slowly during the whole period of motion. As is characteristic, fat	Glycogen levels fall rapidly in the first part of motion. This is so as to provide for the rising

converts slowly to glucose. Notice the overall amount of fat is higher than that of glycogen or adp.	speed in the earlier stages of motion. Characteristically glycogen levels of conversion to glucose are much more faster than of fat.
	
Adp levels also fall rapidly in the first part of motion. Initially though, no activity is registered. This is because glycogen can provide enough energy aerobically for lower speeds. For higher speeds, adp is the main provider because it readily converts without need of oxygen.	The efficiency of energy conversion to atp falls in the beginning stages as well, this indicates that, glycogen is being converted anaerobically at those points, releasing lactic acid, as a by product. Also notice, the efficiency initially is at 75% representing the amount of energy released as heat during conversion to atp, which is waste. This also demonstrates that for the most part of motion, aerobic glycogen contributes for a larger than other energy source.
	
Speed rises sharply and then falls with diminishing availability of atp. For the rest of the period, speed becomes low and constant thereby by putting less pressure on the energy levels.	Distance, also falls much faster in the early stages, and become constant for the remaining time. Note as well, while adp is to credit for the speeds that cause distances to be covered in a short time, it is also demonstrates that it can

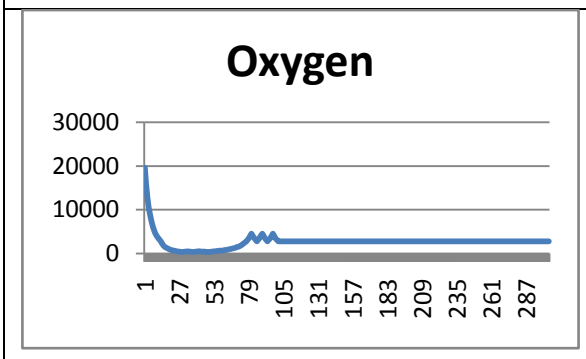
	only sustain this for a short period.
	
Oxygen levels also dramatically fall initially. This is because oxygen follows the relation $oxygen\ \alpha\ \frac{1}{speed^{2}}$. Notice that is affects, the possibility of aerobic conversion of glycogen to atp, hence affecting the overall efficiency of energy conversion.	

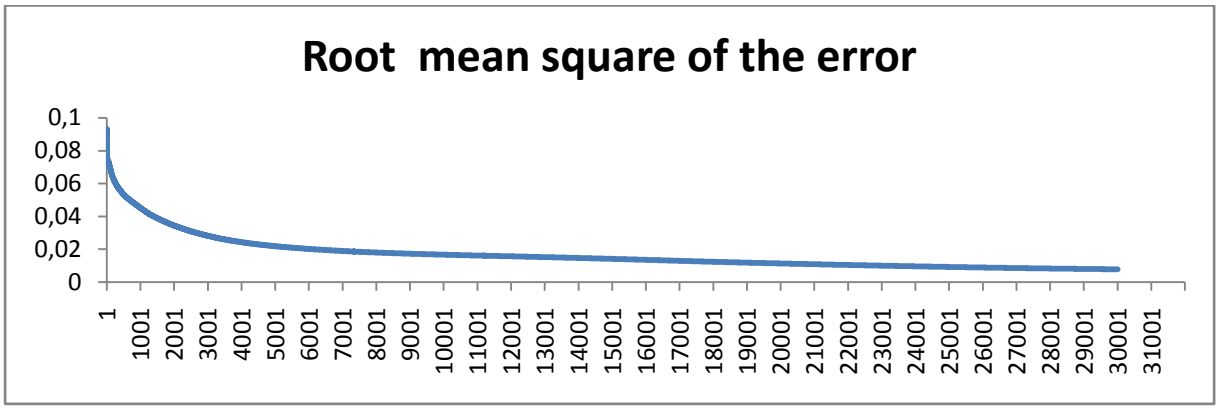
Table 4.2-1 Pacesetter motion attributes in a 300 meters athletic competition

4.3 Q-Learning

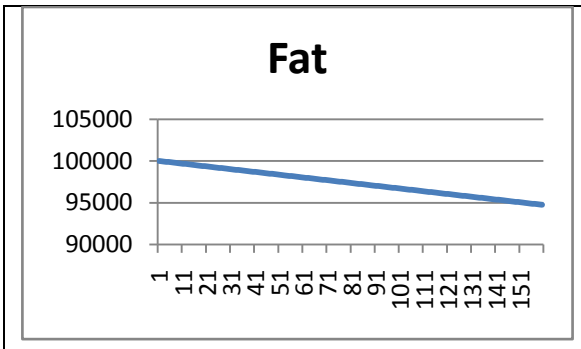
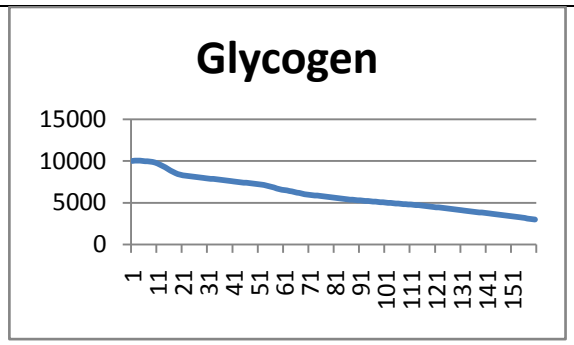
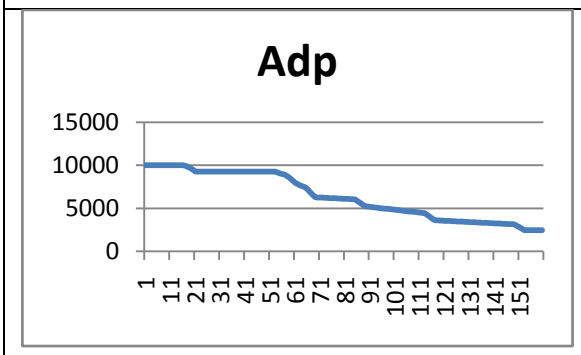
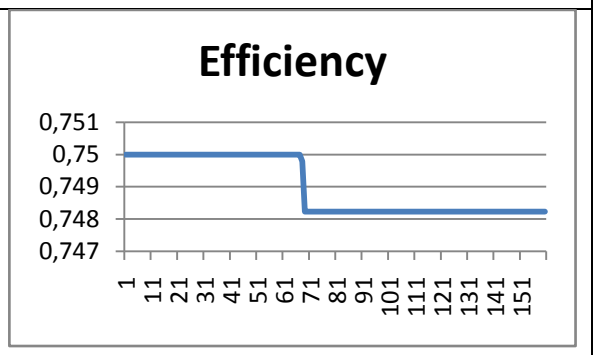
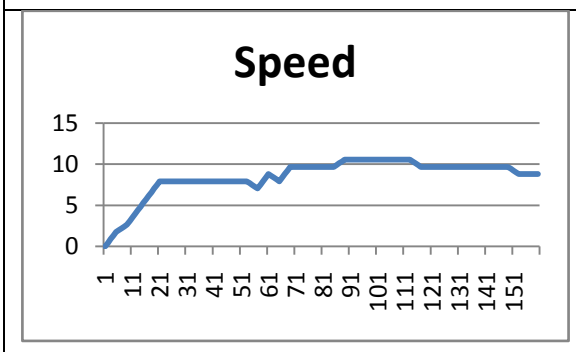
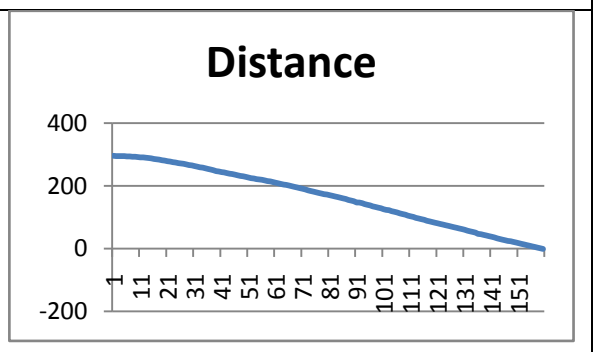
- Training set and strategy

  The AthlecticPriorKnowledge Algorithm introduced in Table 3.4-10: The AthlecticPriorKnowledge Algorithm was used to train the agent. The training consisted of at least 100000 athletic episodes of race covering distance of up to 300m. The Q-table after training contained 96000 unique states.
- The Q-learning for non-deterministic learning Equation 2.3-1 was used for Q-value updates.
- The Cross-validate function verified the performance of the agent. This was done by running 10000 test episodes of up to 300meter race where the Q-learning agent competed against the 2 pace setter agents. A cross-validate was made against the pace settor for 10000 episodes each of a running distance of upto 300meters random episodes. The win ratio for the q-learning agent was 0,8594.

4.3.1 Convergence rate of Q-Learning



4.3.2 Motion and its effect on key attributes

<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Similar trend to pace-setter</td><td>Grandual usage of glycogen as opposed to pace settor</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Grandual usage of glycogen as opposed to pace settor</td><td>Similar to pace</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>The speed choice is look very different from</td><td>The change is distance is more steadier than the</td></tr>
</table>

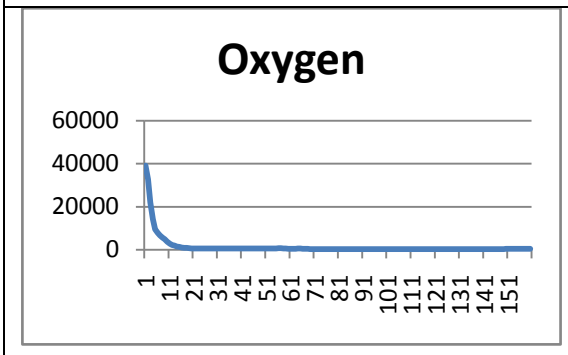
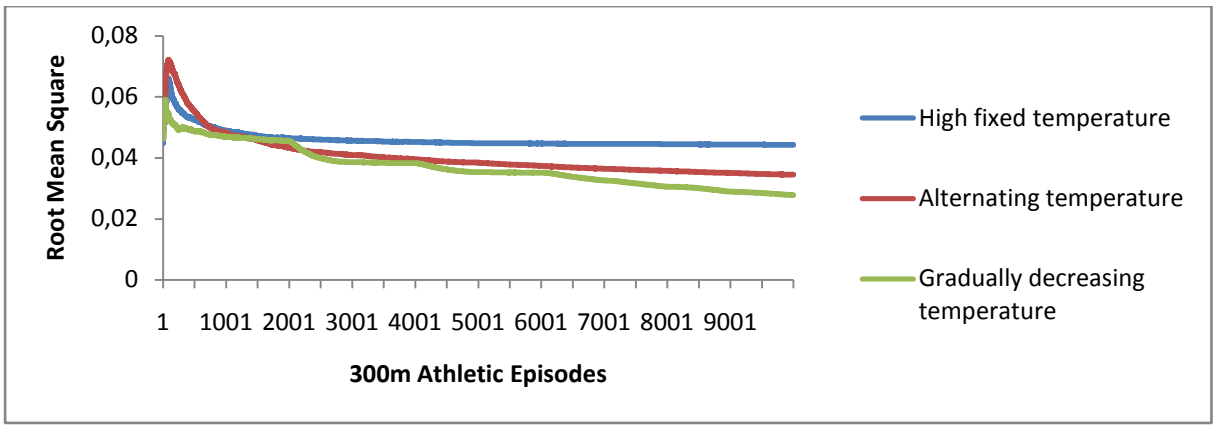
the pace-settor. Top speed are maintain for longer period during the race. An explanation for this could be the acceleleration when maintain at 0 causes less loss of energy	one observed from the pace-settor
	
Nearly the entire more period of motion is anaerobic.	

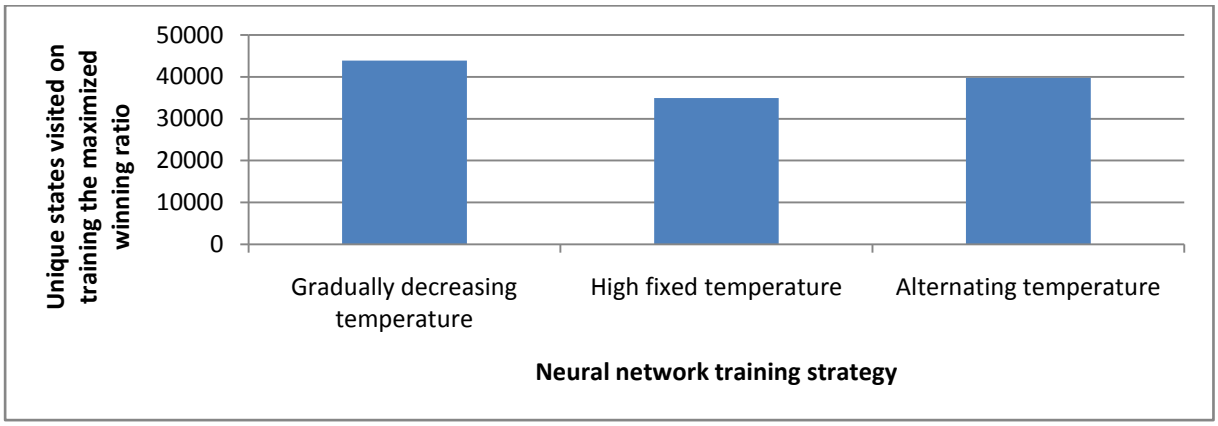
Table 4.3-1 Q-learning motion attributes in a 300 meters athletic competition

4.4 Combined Model

4.4.1 Convergence rate of Q-Learning neural network



4.4.2 Performance of generalizing



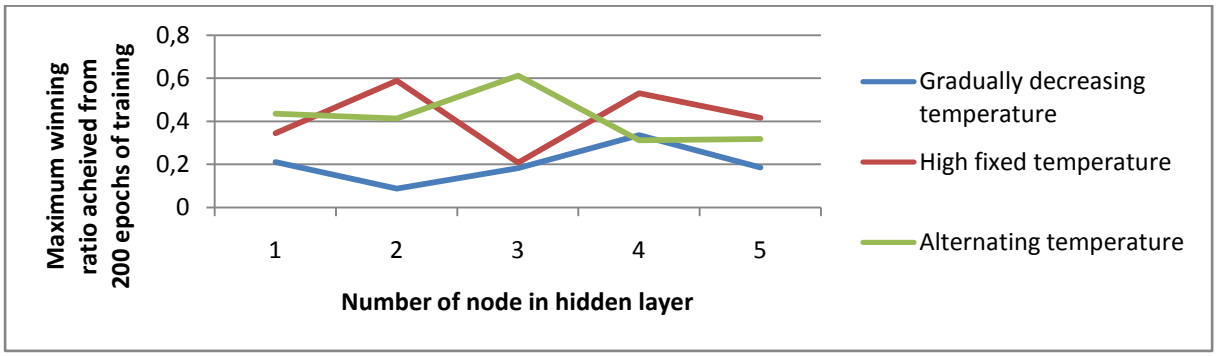


Table 4.4-1 Impact of training strategy on the winning ratio vs. number of nodes in hidden layers

Chapter 5. Discussion

In the empirical section several experiments were performed so to attempt to answer the thesis question. The experiments were based on the athletic competition which was implemented using the theory discussed in Chapter 3.

The experiments were divided into three parts.

5.1 Experiment on the fuzzy rule based agent

The pace-setter agent was implemented to represent current decision-making models for motion. The relevance of the experiment using this agent was to use him as a benchmark to relate of the eventual agent using the proposed on the thesis question. Two experiments were made.

The first experiment, in section 4.2.1, was to reveal the relationship of the agent to the time complexity aspect of the game. The experiment show that the complexity of time increased linearly with every additional agent added to the game. Following the trend it would require at least 1500 agents to reach unacceptable limit.

The second experiment, in section 4.2.2 was intended reveal how motion attributes were affected in the implementation of the athletic competition. The experiments revealed behavior that is consistent with one described in section 2.2.

5.2 Experiments on the Q-learning agent

The Q-learning agent was implemented using the algorithms proposed in section 3.4.3. With amount training described it was able to perform well and win consistently against the pace setter, in various race with random initial configuration. One behavior that was observed in the training of this agent was that it preferred running at high speed with zero acceleration. Constant speeds in general turn out to be cheaper than motion involved changes in acceleration.

5.3 Experiments on the combined Q-learning and neural network agent

´The training of the combined Q-learning and neural network consisted of the same setup as that of the Q-learning, except for the Q-table being replace by a neural network. Training was done using a growing algorithm described in 3.4.7 in order to find the architecture that generalized.

Chapter 6. Conclusion

6.1 Summary of what was achieved

In this thesis, we introduced a model for human motion using realistic muscles energy sources and their usage, realistic sensation and perception for motion, and realistic physics for motion. We then proposed a model for decision-making using a combination of Q-learning and artificial neural networks to see whether it was possible to build an agent that could achieve a long-term goal that required motion given these realistic constraints.

An athletic competition was implemented using this model to assess the performance of the agent. Empirical evidence revealed that, even under complex energy constraints, given enough experience, agents were capable of learning from no prior knowledge to execute motion that achieved long-term goals.

Using Q-learning alone to model decision-making, agents were able to learn and overcome their energy constraints and eventually become consistently competitive. However this model had the shortcomings of potentially crippling game play with worsening space and time complexity.

The combined model of Q-learning and neural network showed signs of learning. It could, eventually learning to reach the finish line of a race consistently. However it did not become as competitive compared to the Q-learning agent. Various reasons could be the cause of weak performance in generalization, including slow convergence, over fitting, convergence bad local minima.

6.2 Contributions of this thesis

This thesis introduced a new solution concept on how realistic human motion can be implemented in games using machine learning techniques which produce adaptable and competitive behavior especially on the part of the non-player character, thus increasing the potential of engaging game play.

In trying to address the thesis question, the thesis also produced a generic model for implementing realistic motion that could be extended to all types of games, and also be used as a framework for implementing motion for non-human characters as well. This model also addressed architectural aspects of modularity and current game design giving it clean interface to plug-in into exist games

6.3 Insights on future work

Realistic motion is not only governed by aspects that were addressed in this issue. As was mentioned in the scope section 1.7, other factors play a role in the decision-making process of realistic motion. The following area could further improve realism of human motion in games

- Metabolism: in this thesis we focused mainly catabolic activities of energy source in motion. However we did not address the replenishment of the energy sources, which is metabolism. By address this issue, a full cycle of the energy process will be complete, and lead to the introduction of more aspect of realism in games, that is, proper diet.
- Environmental aspects: in this thesis we assumed some influence in the form of friction and drag. In real life, the type of surface (such as, ice), humidity, and altitude among play a role in motion. Motion realism can benefit from a proper account of these factors.

Chapter 7. Bibliography

Adenosine triphosphate. (u.d.). Hentede 01. 06 2008 fra Wikipedia: http://en.wikipedia.org/wiki/Adenosine_triphosphate

Athletics. (u.d.). Hentede 19. 05 2008 fra www.olympic.org - official website of the olympic movement: http://www.olympic.org/uk/sports/programme/index_uk.asp?SportCode=AT

Bishop, C. M. (2007). *Neural Networks for Pattern Recognition.* Oxford.

Bishop, C. M. (2006). *Pattern recognition and Machine Learning.* Springer.

Body Atlas (2006). [Film].

Bourg, D. M., & Glenn, S. (2004). *AI for Game Developers.* O'Reilly.

Competitions - Rules and Regulations. (u.d.). Hentede 19. 05 2008 fra iaaf.org International Association of Athletics Federation: http://www.iaaf.org/mm/Document/imported/42192.pdf

Crampton, T. (9. July 2005). *For games, teamwork beats out flash*. Hentede 14. May 2008 fra International Herald Tribune: http://iht.com/articles/2005/07/08/business/ptgames09.php

David Hearn, M. P. (2004). *Computer Graphics with OpenGL.* Peason Prentice Hall.

Drag (physics). (u.d.). Hentede 31. 05 2008 fra Wikipedia: http://en.wikipeida.org/wiki/Drag_(physics)

Friction. (u.d.). Hentede 31. 05 2008 fra Wikipedia: http://en.wikipedia.org/wiki/Friction

Fullerton, T., Swain, C., & Hoffman, S. (2004). *Game Design Workshop.* CMP Books.

Horn, R. S. (u.d.). *Work and Energy in Muscles.* Hentede 19. 05 2008 fra MedBio: http://www.medbio.info/Horn/PDF%20files/muscle_metabolism.pdf

Kleinberg, J., & Tardos, E. (2006). *Algorithm Design.* Addison Wesley.

Matlin, M. W., & Foley, H. J. (1997). *Sensation and Perception.* Allyn and Bacon.

Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill.

Nkya, E. (2008). *Simulating a 5000km run.*

Russel, S., & Norvig, P. (2003). *Artificial Intelligence A Modern Approach.* Prentice Hall.

Sutton S, R., & Barto G, A. *Reinforcement Learning An Introduction.* The MIT Press.