Emotional Agents in Computer Games

Khalil Shihab

***Abstract*— In this paper, we consider emotion as a factor in the decision-making process and actions taken by an agent can be represented by a model, called "emotional model" created with specific focus on computer games development. It is designed to explore people's behavior in certain circumstances, while under specified emotional states. Special attention was given to the thought process and actions displayed in the hypothetical scenarios. We characterized thoughts and actions associated with each scenario and emotional state. Each particular action or proof of steps taken in the thought process was given a percentage value directly proportional to answers given by the test population. Finally, we developed an experimental game program for the evaluation of our emotional decision making model. The aim of the evaluation was to find out how real life agents reacted in certain situations and what processes the human mind runs through when thinking and acting upon certain situations.

***Keywords*—**Emotional Model, Computer Game, Evaluation, Intelligent Agents

I. Introduction

COMPUTER Game Software (CGS) has become increasingly popular. Unlike before, today's games are geared toward an older demographic and as a result they have become much smarter and more complex. Players are constantly looking for challenging CGS and this is can be achieved due to the recent advances in Artificial Intelligence [1].

Over the last five years, games have become increasingly intelligent and intellectually demanding [2]. If we compare an older game to any of the current generation games, it will become apparent that these new games are much more difficult to play. Opponents in these games have also become smarter and now seem to exhibit what could be considered intelligent behavior. Some games even have agents that learn, to a certain degree, and adjust their decisions accordingly, even cooperating against you though even at this stage, they are by no means perfect.

Agents still seem to exhibit strange behavior, such as walking into walls and using items inefficiently. Even though, to a certain degree, agents currently seem to act in an intelligent way and make intelligent decisions, there is still something lacking in their behavior. Their actions are although intelligent still seem quite robotic. Therefore, this work addresses this area of study.

This paper introduces emotion as a factor in the decision-making process and actions taken by an agent. Human emotions play a large part in how an individual thinks and acts. For example, decisions made in anger can often be different from those made otherwise. Likewise, trying to perform an action like throwing a ball can also be affected by the mood an individual is in, which is governed by emotions. Emotions can be a driving force behind the types of decisions and actions and individual makes [3]. Depending on ones emotional state, the individual can make better or worst decisions and perform action more or less effectively [3, 4, 5]. Therefore to bring artificial intelligence to the next level, that is closer to human, emotions need to be incorporated in the decision-making process and actions of agents. If agents can be made to behave with emotion then they will appear more human, which is exactly what is wanted (computer controlled agents simulate a human opponent).

Adopting this emotion approach to agents, artificial intelligence may not always result in an optimal decision or action [6, 7, 8]. Rather it will result in the best possible decision or action given the agents emotional state. Human players get angry, nervous and frustrated and this affects the way they play. This should be no different for computer controlled agents as the aim of this thesis is the development of an agent that exhibits human like behavior, mistakes and all.

II. Background

Artificial intelligence (AI) has been growing and maturing in the passing years and the domain of video games has become an increasingly popular platform for artificial intelligence research [9, 10]. As games become more complex and realistic, so too does the AI that drives these games. Games may be simplified when compared to the real world but none the less they provide complex, dynamic environments and situations which even human players find challenging. Although AI in videogames has been constantly improving, it is still at a stage where inflexible and predictable behavior is exhibited.

A. Goal and resource using ArchitecturE (GRUE)

Gordon and Logan [8, 9] have proposed GRUE, which is a new architecture that aims at improving these weaknesses. It

K. Shihab is currently an Associate Professor in the School of Computing & Design, Swinburne University of Technology (Sarawak Campus), Jalan Simpang Tiga, 93350 Kuching, Sarawak, Malaysia,Tel: 60 82 416 353 Ext: 7718, Mobile: +60 14 9903139, Fax: +60 82 423 594, email: kshihab@swinburne.edu.my.

uses teleo-reactive programs (TPRs) which basically consist of a series of rules, with each rule containing some number of conditions and actions. Running a TPR, it evaluates all the rules and executes the actions of the first rule whose conditions evaluate to true when compared to the world model that is stored in the agent's memory. The resulting actions can be said to be durative as they carry out as long as its conditions are true. In this architecture the agents use TPR to pre-define plans for achieving goals. Furthermore it is here that multiple actions are allowed to be executed during each cycle.

Game agents may encounter situations where several items may be adequate in achieving a task or where objects come in quantities such as money and ammunition. GRUE is designed specifically for these types of situations and is built around the key concept of resources.

GRUE allows the game agent to generate new top-level goals depending on the current game situation and assign priorities to these goals based on the current situation. For example, an agents goal may be attacked, but if it is injured it may then generate a new goal which would be to heal itself before continuing with previous goal of attacking. Here the goal of healing would be given a higher priority and the first goal of attacking would be given a lower priority. Once the agent has carried out the goal of healing it will then continue with the original goal.

Multiple tasks can run actions in parallel during each cycle when it is possible. If the agent's task is to search for ammunition then actions needed to carry out this task can be run in parallel during each cycle. Actions may include searching, defending, attacking or healing when hurt and so forth.

A complete GRUE has been implemented for the Tileworld environment and the agent performs well, demonstrating that resource use with preferred properties is advantageous. A basic GRUE has also been implemented for the Unreal Tournament game and performs less impressively showing predictable behavior. However, the authors do believe that a complete GRUE agent will perform much better.

B. The use of influence diagrams (IDs)

In recent years, game theory and decision theory have had a profound impact of artificial intelligence in video games [11, 12]. Traditionally, multi-agent systems using game-theoretic analysis for decision making use a normative approach [2]. It is here that decisions are derived rationally from the game description. However, this approach is believed to be insufficient and it does not capture the decision making process of real life agents. Real life agents (real people) may be partially irrational or may use models other than the real world (the game model) to make decisions [8]. Also agents may be unsure about their opponents' decision-making processes. Network Interface Diagram (NID), developed by Gal and Pfeffer, allows for situations in which agents have an incorrect mental model of how the world works and also allows for instances where a modeler has uncertainty about another agents model.

The basic building blocks of a NID are influence diagrams (IDs). IDs consist of a direct graph with three types of nodes as described below:

- Chance nodes – drawn as circles and represent random variables.
- Decision nodes – drawn as rectangles and represent decision points
- Value node – drawn as diamonds and represent the agent's utility which is to be maximized

C. Multi agents' coordination

Multi agents' coordination is another important area in video game Artificial Intelligence. In many of today's games computer controlled agents must work together in an intelligent and believable way against the human player. In multi agent coordination, the aim is to find a satisfactory solution that is fair, stable and optimal to all agents. In human society this often involves a trusted third party in the negotiating process among all agents to insure that all agents should cooperate and are committed [14, 16]. As with most Artificial Intelligence problems, this too will be modeled to work in the same way as the real world.

Wu and Soo [13] described how a trusted third party can be involved in the negotiation of multi agent coordination to deal with many difficult and challenging game situations.

Axelrod and Genesereth [10] showed that rational agents are able to coordinate and cooperate with a game theoretical deal-making mechanism even without communication.

D. The Emotional Decision Making Model

For our emotional decision making model to work and mimic realistic human behavior, we developed an experimental model. We wanted to find out how real life agents reacted in certain situations and what processes the human mind runs through when thinking and acting upon certain situations.

As shown in Figure 1, there are seven key stages in the 'emotional decision making model'. These are numbered one through to seven respectively. Note that these numbers do not represent the process order or direction of navigation, rather, they are nothing more than identifiers which will aid us in the explanation of each of the parts that collectively make up the emotional decision making model.

We begin at point (1), the game agent. The game agent represents any computer-controlled entity. This can be anything from an animal to an opposing character. In other words, a game agent is any 'thing' that is not controlled by the player. This game agent will, at any given time, be in an emotional state. Depending on this emotional state, the agent will make a decision, which will trigger an action. This action can then further affect the agent's current emotional state, therefore changing it. This process is recursive, in that it is continually cycling and constantly changing until the game agent ceases to exist.

Moving on to point (2), we have the game agent's emotional state, referred to as 'emotion'. It is here that the agent's current emotional state is stored, which will continue to change as the game progresses and the game agent makes decisions and performs actions. Actions performed by the game agent will be influenced by the emotion. This will be covered in greater detail throughout point (4). Note: that a game agent is in an emotional state at any given point in time, thus it is considered the heart and soul of this model.

Next we have point (3), referred to as 'decision'. Here with game agent will store all possible decision available while under a particular emotional state. The decision with the highest percentage value will always take precedence over decisions with lower percentage values, which will be executed. If there are two or more decisions with equal percentage values, the first decision in the list out of the possible decisions will be selected and executed. Decisions are stored as a list and are traversed until a suitable decision is found. Let us set up a scenario to illustrate the mechanics of this step. This scenario will require the game agent to decide on weather to attack an overwhelming opponent, or retreated from battle. Possible choices available to the game agent are referred to as 'decision candidates' each decision has a percentage or weight attached to it. The game agent is in a scared emotional state. Bellow is the agent's possible decisions that correspond to the emotion it is in.

```
Decision Candidates
Attack 10%
Retreat 90%
```

As retreat has the highest percentage value the game agent's decision will be to retreat from battle. This decision opens up possible actions that the agent may execute, which will be covered in the next section. Note that once a decision has been made 'decision candidates' are cleared in preparation for the next iteration. It is important to note that this is a simplified accounting of how this section of the decision making model works. Sub decisions may be needed to properly select the best course of action. For example, health remaining, distance and so on could be taken into consideration, though this paper will only cover simple, non-nested decisions.

Point (4) is referred to as 'action' and works in much the same way as point (3). It is important to note at this time that possible actions are provided by the decision selected. Here all possible actions available to the game agent will be stored in a list and the action with the highest percentage value will take precedence over lower valued actions. As before, actions with equal percentage values will be selected using the first-on-list method. As with 'decision', 'action' acts in much the same way, in that a list of possible actions are provided and selected based on their percentage value. Possible actions available are provided by the decision made. Note that in this step the emotional state of the game agents is no longer relevant. This is because the previous step, 'decision' was carried out while under the influence of 'emotion' and thus the possible actions provided to 'action candidates' will follow suit. It is important to remember that decisions made under the influence of a particular emotion will always lead to actions made corresponding to that emotion. In other words, decisions made in anger will lead to actions performed in anger. Below are possible 'actions' provided by 'decision'.

```
Action Candidates
Retreat to area occupied by friendly
game agents 40%
Retreat to nearest safe location 30%
Retreat to Base 30%
```

As in the above example the choice with the highest percentage will be fired and 'actions' will be cleared in preparation for the next iteration.

Now we reach what are known as 'outside effectors'. Point (5) is the first of these and is referred to as 'game environment'. During a game many things are simultaneously happening. Not only are the player and game agents performing actions that affect one another, but the game environment is constantly changing and also affecting the game agent. The game environment can be anything from rain in a game to a particular geographical stage structure, each of which will trigger selected emotions in the game agent, see Example 1.

Next, we reach point (6) referred to as 'prior actions/decisions'. Here previous actions that may trigger particular emotions are stored. Once a game agent makes a decision and performs an action, often, the action performed may trigger further emotional states. Again this will be explained in greater detail in Example1.

Finally, we reach the final point in the emotional decision making model, point (7). This is referred to as 'other agent's actions/decisions'. Many times in a game, there will be multiple game agents controlled by the computer. These agents will most likely interact with each other, thus having an effect on one-another's emotional states. This allows for realistic teamwork and quarrels between game agents (i.e. if agent accidentally shoots team member, team member may fire back in anger).

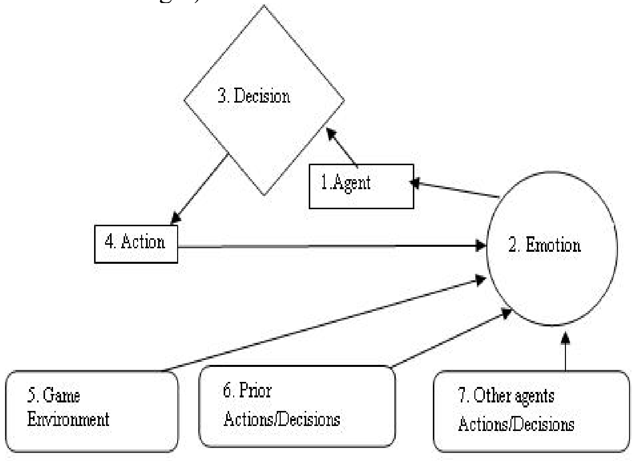

Fig. 1. The emotional decision making model

III. Model Simulation

For our emotional decision making model to work and mimic realistic human behavior [12], we developed an experimental model. We wanted to find out how real life agents reacted in certain situations and what processes the human mind runs through when thinking and acting upon certain situations [13, 14, and 15].

A. Experimental game

In this experiment we used two agents, namely A_1 and A_2 that simulate the human reasoning process. When people reason about the behavior of others they often express their emotion (i.e., feeling sorry for someone, feeling happy for them, resenting their good fortune, or gloating over their bad fortune). To do this, agents maintain a list of cases establishing points of view of other agents and use these cases to take future actions. These two agents work in a cooperative environment in which the agent A_2 should wait for a message from the agent A_1 in order to take the right step in the right direction.

The agents described in this experiment are able to participate in a multi-stage game in which one intelligent agent (A_1) observes and interacts with a naïve agent (A_2) express feelings about other agent's actions. The naïve agent uses those emotions to take the right action. These emotions are vital to the decision-making process and to manage competing motivations.

Our naïve agent can learn through the feedback from the intelligent agent. The agent can pass one room to another but has no knowledge of the environment. It does not know which sequence of doors the agent must pass to go outside the building.

The game environment for the intelligent agent (A_1) is a simple evacuation of an agent from any room in the building, see Figure 2. At the start of the game, the agent is allocated to Room C (initial state) and we want the agent to learn to go outside the house (F). At each landmark (initial state, obstacle and destination state), A_1 should update the shared information.

We consider each room (including outside the building) as a state. The agent's movement from one room to another room is called an action. Figure 3 shows that states are represented by nodes in the state diagram, while actions are represented by the arrows.

From state C, the agent can go to state D because state C is connected to D but with reward zero because D is not the goal state. From state C, however, the agent cannot directly go to state B because there is no direct door connecting room B and C (thus, no arrow). From state D, the agent can go either to state B or state E or back to state C (look at the arrow out of state D). If the agent is in state E, then three possible actions are to go to state A with reward zero, or state F with reward 100 (because F is the goal state) or state D. If the agent is in state B, it can go either to state F or state D. From state A, it can only go back to state E.

The agents described in this experiment are able to participate in a multi-stage game in which one agent (A_1) should learn through experience without a teacher by applying the Q-learning technique, which is a reinforcement learning technique that bridges the gap between supervised and unsupervised learning categories. At the start, the agent can pass from one room to another but has no knowledge of the environment. It does not know which sequence of doors the agent must pass through to go outside the building. After a sequence of training sessions, the agent should be aware of the environment and the location of the target and other significant points. A map like for all of these pieces of information should be produced and saved in a sharable location to allow both robots easy access, see Figure 2 and Table 1.

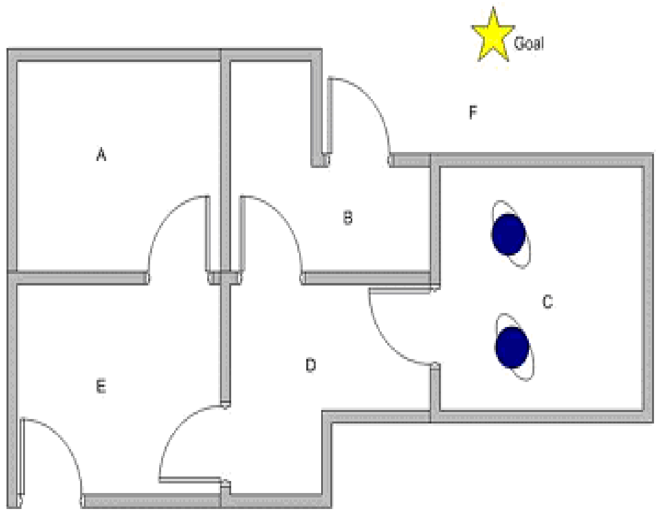

Fig. 2 a simple house evacuation

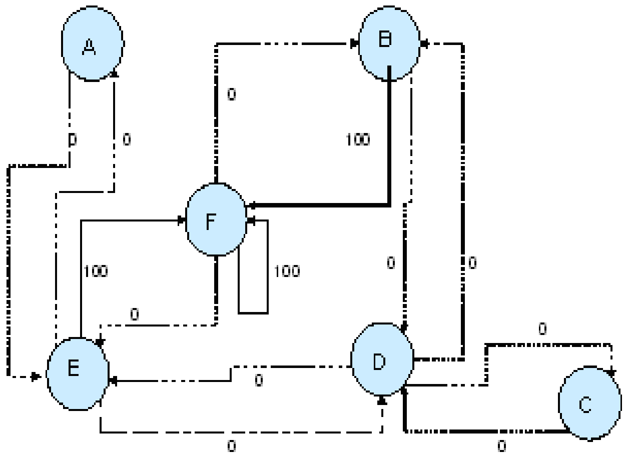

Fig. 3 the state diagram

Table 1. State reward values

	Action to go to state					
Agent in state	**A**	**B**	**C**	**D**	**E**	**F**
A	-	-	-	-	0	-
B	-	-	-	0	-	100
C	-	-	-	0	-	-
D	-	0	0	-	0	-
E	0	-	-	0	-	100
F	-	0	-	-	0	100

Table 2: Distances between landmarks, where N=North, S=South, E=East, W=West, NE=North East, SW=South West, SE=South East

	B	C	D	E	F
A	20/E	40/E	20/SE	20/S	40/NE
B		20/E	30/S	50/SW	20/N
C			20/SW	60/SW	50/NE
D				20/SW	70/N
E					50/NE

Q-learning requires a similar matrix name Q in the brain of our agent that will represent the memory of what the agent has learned through many experiences. The rows of matrix Q represent the current state of the agent, the columns of matrix Q point to the action to go to the next state. As mentioned above, the agent starts without knowledge of the environment, thus we put Q as a zero matrix. The transition rule of this Q learning is the following simple formula

Q(state, action)=R(state, action)+α. Max[Q(next state, all actions)] (1)

Where,

Q is the transition matrix; rows represent states and columns represent actions,

R is the instant reward matrix, and

α is the learning parameter.

B. *Q-Learning Technique*

```
Given: State diagram with a goal state
(represented by matrix R)

Find: Minimum path from any initial state
to the goal state (represented by matrix
Q)

Q Learning Algorithm goes as follow:

   1. Set parameter α, and environment
      reward matrix R
   2. Initialise matrix Q to zero
   3. For each episode:
    o    Select random initial state
    o    Do while the goal state not
         reached
       ▪   Select one among all possible
           actions for the current state
       ▪   Using this possible action,
           consider to go to the next
           state
       ▪   Get maximum Q value of this
           next state based on all
           possible actions using formula
           (1)
       ▪   Set the next state as the
           current state
      End Do
```

The above algorithm is used by our intelligent agent (A1) to learn from experience or training. It was proposed for Markovian's problems decision, with discrete states and actions spaces. The Q-Learning is, therefore, well suited for on-line applications that are characterized by discrete states, and generally performs well in practice. Each episode is equivalent to one training session. In each training session, the agent explores the environment (represented by Matrix R), gets a reward (or nothing) until it reaches the goal state. The purpose of the training is to enhance the 'brain' of our agent that is represented by the Q matrix. The Q-learning algorithm is guaranteed to converge to Q*(s, a), the optimal action value function, with probability 1 as long as each state-action pair is continually updated [12, and 13]. The Q-learning algorithm learns an action value function, which is the expected sum of discounted future rewards for taking action a in state s and behaving optimally thereafter. This action value function is often represented in a Q-table of the form Q(state, action), which is updated during the learning process, see Table 1. More training will produce a better Q matrix that can be used by the agent to move in an optimal way. In this case, if the Q matrix has been enhanced, instead of exploring around and going back and forth to the same room, the agent will find the fastest route to the goal state.

Parameter α is in the range 0 to 1(0<< α<<1). If α is closer to zero, the agent will tend to consider only an immediate reward. If α is closer to one, the agent will consider a future reward with greater weight, and be willing to delay the receipt of a reward.

To use the Q matrix, the agent traces the sequence of states, from the initial state to the goal state, hence producing a trajectory of its path. At each step of the sequence, the agent should find an action that maximizes Q for the current state. The intelligent agent was trained in the above mentioned environment, see Figure 3. The training session started with α=0.25.

C. XML Map

Once the path from the initial state to the goal state is found, the intelligent agent (A_1) uses the distance table (Table 2) to produce an XML map for future navigation and for coordinating the movements of the naïve agent (A_2). In addition to the name and the description of each state, the map should have the distance from a current state to the next state in the navigation path.

Using the above mentioned game environment, the intelligent agent produces the following direction and XML maps.

Forward direction map (C -> D -> B -> F) with its calculated distance d=20+30+20=70

Backward direction map (F -> B -> D ->C) with the same distance d=70

```
  <?xml version="1.0"?>
  <ENVIORNMENT>>
   <LANDMARK  “ID”=1>
      <CODE> C </CODE>
      <EDGE TO “ID”=2 LENGTH=20
                 DIRECTION=”SW”>
      <DESCRIPTION>Initial State
         </DESCRIPTION>
    <URL>Obj1.jpg</URL>
    </LANDMARK>
  <LANDMARK “ID”=2>
    <ID>2</ID>
    <CODE> D </CODE>
    <EDGE TO “ID”=3 LENGTH=30
                 DIRECTION=”NW”>
      <DESCRIPTION>Obstacle
            </DESCRIPTION>
    <URL>Obj2.jpg</URL>
    </LANDMARK>
<LANDMARK “ID”=3>
    <ID>2</ID>
    <CODE> D </CODE>
    <EDGE TO “ID”=4 LENGTH=30
                  DIRECTION=”N”>
      <DESCRIPTION>Obstacle
        </DESCRIPTION>
    <URL>Obj2.jpg</URL>
      </LANDMARK>
<LANDMARK “ID”=4>
  <CODE> F </CODE>
      <EDGE TO “ID”=4”  LENGTH=0
                  DIRECTION=”T”>
    <DESCRIPTION> Target State
                  </DESCRIPTION>
    <URL>Obj3.jpg</URL>
    </LANDMARK>
  </ENVIRNMENT>
```

The XML map documents should conform to the following document type declaration:

```
  <!ELEMENT map (node*, location*,
  metalocation*)
  <!ELEMENT node (edge+)>
  <!ATTLIST node id ID
              #REQUIRED>
  <!ELEMENT edge (subnode*)>
    <!ATTLIST edge to IDREF
               #REQUIRED
          length CDATA #REQUIRED
          direction
      (N|S|E|W|NE|NW|SE|SW|T) #IMPLIED>
  <!ELEMENT subnode EMPTY>
     <!ATTLIST subnode position
               CDATA #REQUIRED>
<!ELEMENT location (#PCDATA)>
   <!ATTLIST location id ID
     #REQUIRED node IDREF #REQUIRED
        category IDREFS #REQUIRED>
<!ELEMENT metalocation (#PCDATA)>
<!ATTLIST metalocation id ID #REQUIRED
sublocations IDREFS #REQUIRED>
```

D. Algorithm used by the naïve agent

Input a list of emotions that produced by the intelligent agent (during the first play, the list is empty) and go through the following steps:

```
      1. Set  current  state  =  initial
         state.
      2. From current state, move to the
         next state.
      3. Add the emotional expression of
         the  intelligent  agent  to  the
         list.
      4. Set current state = next state

  Go to step 2 until current state =
goal state
```

The algorithm above returns a list of sequence of states and their associated emotional expressions from initial state until goal state. For the next game sessions, the naïve should use the list of emotions that is produced by the first play to make its future moves.

IV. Using Inter-Process Communication Sockets

Sockets are the most popular form of the Inter Process Communication (IPC) protocols. Network applications use sockets to communicate over a TCP/IP networks. A socket is one end-point of a two-way communication link between two programs running on the network. Because a socket is bidirectional, data can be sent as well as received through it.

Our Agents send and receive messages from each other and these messages should match with the time calculated by each agent. This time is based on the distance and the time taken to finish the job. The naïve agent (A_2) should send its position to the intelligent agent (A_1) at constant intervals of time and at landmark points; these are the point of origin (the initial state), the vertices, and the target. Also, A_2 should send messages to A_1 at critical events. These events are as follows:

- `task start (task_id, type)`
- `task completion (task_id, type)`
- `task failure (task_id, error message)`

Once the intelligent agent is trained, using the Q-Learning technique and the time intervals are recorded at each landmark point, the task environment becomes known. Each agent is capable of detecting the current position of the other agent and the position of the point of origin (the initial state), and the target, thus allowing the robots to maintain collision avoidance while they are moving, see for example [2, 13].

During the training sessions, the intelligent agent produces XML map that is used to navigate through the search space. As stated before, the movement of the naïve agent is known to the intelligent agent through the shared information that produced by the IPC mechanism. Also, the estimated time required for the naïve agent to move from one landmark (a spot or a target) to another is calculated and updated to reveal the current position and the estimated time required to reach to the next landmark, i.e. the naïve agent is required to send a message to the intelligent agent at each landmark. If for some reason that the naïve agent does not reach to the next landmark or a message is not received according to the anticipated time, a halt command (task failure) is issued to stop the process.

Many functions and programming codes are implemented in order to accomplish the experiment. Some of these functions are as follows:

Synchronize function: It is used to synchronize time and organize the movement of the robots. If one of the stationary robots is busy with a task, the other robot should wait until the task is finished.

The pseudocode of this function is as follows:

```
Timer() = current time or clock time
Time_needed= the time needed to carry out
the task at hand

endTime = Time() + Time_needed
do while endTime>Time()
loop
```

Winsock (Window Socket) Function: It defines a network programming interface for Microsoft Windows which is based on the "socket" paradigm popularized in BSD Unix. It encompasses both the familiar Berkeley socket style routines and a set of Windows-specific extensions. It is used here to allow smooth communications between programs that control the robots using the server/client paradigm.

Start Function: It reads a user option. It first checks the status of the robots and issues a function call to the synchronize function if the required robot is not either in the busy or the wait state. After assigning the task to a robot, it should send a message to all.
The pseudocode is as follows:

```
Check if a robot x = true and connection =
true;
Then if user option = true;
Then if user selection = true;
   Then call the synchronize
       function;
    Call WinSocket();
    Send a message to the robots;
      Else send error-message “Error user
      selection”;
   End;
     Else send error-message “Error
         user option”;
End;
Else send error-massage “No connection”;
End;
```

In order for the naïve agent to play a better game, it keeps the emotional expressions of the intelligent agent in a list named emotions. For each position in the game, that is, for each possible move, an emotional expression is added to the list representing the emotional state of the intelligent agent. When the naïve agent starts, the value of every entry in the list of emotions is initialized to zero, corresponding to the absence of any feedback from the intelligent agent. After each move, the naïve agent examines the emotional expression of the intelligent agent.

V. Conclusion

This paper has attempted to address the possibility of incorporating emotion in game agents. It begins by proposing a model, the emotional decision making model, and then applying emotional data to drive our emotional decision making model. Emotional data was gathered via an emotional questionnaire aimed at identifying particular decisions and actions made under certain emotional states. The emotional states explored were the ‘normal’ emotional state and anxiety. The results of these were then studied and scrutinized and emotional traits were identified. The results achieved by the questionnaire were then applied to the emotional decision-making model and examples of it in action were explored.

Future research in the field of emotional decision making for game agents could span into a various directions.

Firstly, a thorough analysis of decisions and actions while under particular emotional states could be carried out. This paper only addresses four emotions. There are countless emotions, all present in our everyday lives that would benefit gaming and game agents.

Secondly, a gender specific study on emotional characteristics and emotional transitions could be carried out. Males may be more susceptible or more likely to show evidence of particular emotional characteristics as opposed to females and vice versa, therefore doing such a study would help improve the believability of the game agent's decisions and actions.

Thirdly, the emotional decision-making model could be integrated with other decision-making models. This paper considers 'emotion' as the key factor in decision making, however in reality there is a number of key factors that are involved. By combining these, a more realistic and advanced decision-making model could be developed.

Finally, the emotional decision making model could be implemented in a game situation. The model and data are already available, implementation would further confirm the ideas explored in this paper.

REFERENCES

[1] Bererton C., State estimation for game AI using particle filters. In: D. Fu, S. Henke and J. Orkin, Editors, Challenges in Game Artificial Intelligence: Papers from the 2004 AAAI Workshop, AAAI Press, Menlo Park, CA (2004), pp. 36–40 Technical Report WS-04-04.

[2] Fielding D., M. Fraser, B. Logan and S. Benford, Reporters, editors and presenters: Using embodied agents to report on online computer games. In: N.R. Jennings, C. Sierra, L. Sonenberg and M. Tambe, Editors, Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004) **vol. 3**, IEEE, New York , pp. 1530–1531, 2004.

[3] Orkin J., Symbolic representation of game world state: Toward real-time planning in games. In: D. Fu, S. Henke and J. Orkin, Editors, Challenges in Game Artificial Intelligence, AAAI Press, Menlo Park, CA (2004), pp. 26–30 Technical Report WS-04-04.

[4] El Rhalibi A., Nick Baker and Madjid Merabti, Emotional agent model and architecture for NPCs group control and interaction to facilitate leadership roles in computer entertainment, ACM International Conference Proceeding Series; Vol. 265, Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, Pages: 156 – 163, 2005.

[5] Chang, Yi-Hsing, Lu, Tsung-Yi, Fang, Rong-Jyue, A validity E-learning system based on knowledge management and intelligent agents, *WSEAS Transactions on Systems*, Vol. 6, No. 17, 2007, pp. 1297-1309.

[6] Hunyadi D.,Pah L. and Chiribuca D. A Global Model for Virtual Educational System, *WSEAS Transactions on Information Science and Applications,* Vol 6, 2009, pp. 374-383

[7] Shihab K. Performance Tuning of Novell Netware Based on Fuzzy Reasoning, *International Journal of Computers*, Issue 1, Volume 2, 2008, pp. 80-88, 2008.

[8] Gordon, E., and Logan, B., Managing goals and real world objects in dynamic environments. In Davis, D., ed., Visions of Mind: Architectures for Cognition and Affect, 2004.

[9] E. Gordon and B. Logan, Game over: You have been beaten by a GRUE. In: D. Fu, S. Henke and J. Orkin, Editors, Challenges in Game Artificial Intelligence: Papers from the 2004 AAAI Workshop, AAAI Press, Menlo Park, CA (2004), pp. 16–21 Technical Report WS-04-04.

[10] Robert Axelrod, The Evolution of Cooperation, Basic Books, Inc., New York, 1984.

[11] O'Brien, J., A flexible goal-based planning architecture. In Rabin, S., ed., AI Game Programming Wisdom. Charles River Media. 375–383, 2002.

[12] Freeman D., Creating Emotion in Games, New Riders Publisher, ISBN: 1592730078, 2004.

[13] Wu, S. H. and Soo, V. W., Game Theoretic Approach to Multi-Agent Coordination by Negotiation with a Trusted Third Party, In Proceeding of the Third International Conference on Autonomous Agents, 1999.

[14] Wooldridge M. An Intoduction to MultiAgent Systems, 2ed, John Wiley & Sons, 2009.

[15] Parra C, Colina E, Chacón E, Intelligent Supervisory Control Design Framework for Fault Exposed Processes. NAUN InternacionalJournal of Circuits, Systems and SignalProcessing, Vol 1, Issue 3, 2007, pp. 251-258.

[16] Magdy Saeb, M and Fathy, C. Performance Evaluation of Mobile Agent-based Dynamic Load Balancing Algorithm, WSEAS Transactions on Computers, Issue 3, Vol. 2, 2003, pp. 811-819.