Go*Team, an Instance of the Simulation Framework

Jagiello, J. [1], Eronen M. [1]

[1] Defence Science and Technology Organisation, Canberra, Australia
Email: jerzy.jagiello@dsto.defence.gov.au, marko.eronen@dsto.defence.gov.au

Keywords: *simulation, framework, software, gaming*

EXTENDED ABSTRACT

Network-Centric Warfare (NCW) is a new military doctrine or theory of war that seeks to translate an information advantage into a competitive war fighting advantage through the robust networking of well informed geographically dispersed forces (Alberts *et al*. 1999, http://en.wikipedia.org/wiki/Network-centric_warfare).

The focus of NCW efforts has largely been concentrated on issues related to technology and infrastructure. However, there is a growing trend in the scientific community to analyse the human aspect of network warfare. The human factors community has concerns about the impact of technology on human performance, and has identified a need for investigation of individual and group behaviours in a NCW context.

There are many possible ways of testing hypotheses regarding human and organisational behaviour. A practical approach is to conduct a real life experiment. Sometimes it is impossible to set this up due to technical and economical constraints. It can also be difficult to establish validity of conclusions based on a limited experimental sample. An alternative is to abstract some of the key attributes of a real life system into a model, allowing for unlimited repetition and an understanding of the behaviour of the real life system. The computerized Go*Team game is an example of such an approach. Go itself has nothing to do with NCW per say, but it creates an opportunity for simulating cooperation and coordination between teams and individuals. By creating a competitive and collaborative environment where players/teams compete against each other, human factors may be identified which could have a profound impact on the outcomes of future NCW wars.

Go*Team is based on the traditional Go game in which players place black and white stones onto a board in order to occupy territory on the board (http://www.britgo.org/intro/intro1.html).
Go*Team is designed to allow a number of competing teams to play Go with a number of players in each team. Every player on a team has a local view of the game. Players on the same team must collaborate if they want to have a more complete picture of the actual game state.

The Go*Team game was implemented as an instance of the Simulation Framework (Jagiello *et al*. 2007). The core of the framework provides the software infrastructure for storing and distributing the simulation state across the network. This paper describes how the Go*Team requirements were transformed into the framework vocabulary and constructs.

1. GO*TEAM REQUIREMENTS

The original Go game consists of black and white stones and a square board with grid lines. The board sizes can vary, with a standard board having 19x19 grid lines. A Go game requires two players to take turns placing stones onto the grid line intersections of the board. The object of the game is to occupy territory on the board. At the start of the game, players place stones onto the board staking claim to areas which they intend to occupy. As the game progresses, players have to defend their positions while attempting to gain more territory. Stones cannot be moved once they are placed onto the board. However they can be captured resulting in their removal from the board. Stones are captured when they are surrounded by an opponent's stones. While capturing stones is not the object of the game it does provide a useful way of gaining additional territory. The winner of the game is the player who controls a larger proportion of the board when the game ends. The Go*Team game is a modified version of the original Go game adapted with the purpose of simulating an NCW environment. The Go*Team allows more than one person to play for a particular stone colour, and the stone colours are not limited to just black and white. People playing for the same colour are on the same team and they cooperate with each other to achieve victory for their team. Teams can form alliances to simulate coalition forces. A game can be played by many teams on many boards with a limited number of allocated stones. An important element of Go*Team is the limited visibility of the game state. The individuals from one team only have a local view of the game. They cannot see where the stones of other players on their team are, and they cannot see opposition team stones that are not close to their own stones. However it is possible for a team to "reconstruct" a complete picture of the game state if everyone on the team shares their knowledge.

2. GO*TEAM MODEL

Having in mind the Go*Team requirements and the Simulation Framework architecture the Go*Team implementation model can be represented as depicted in Figure 1. The input $\mathrm{Pr}eAction(k\Delta T_k)$ and the output (SeverAction, GameSataus, Teams, Allies, IllegalMoves, Winners) variables inside the repository store are manipulated by the players and the rule modules. Here is the outline of the input/output variable structure.

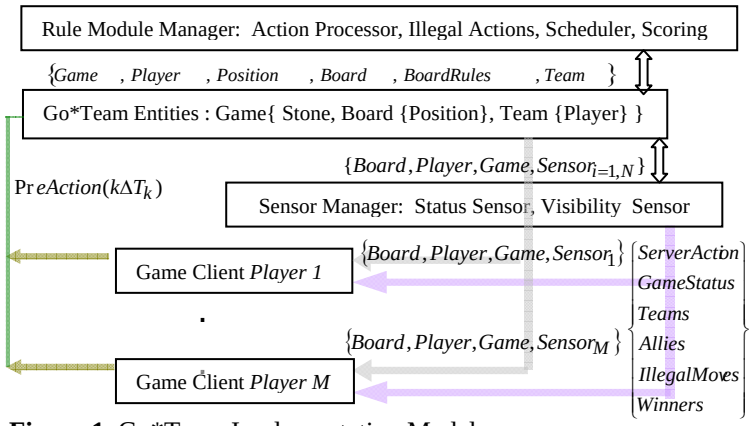

Figure 1. Go*Team Implementation Model

$\mathrm{Pr}eAction(k\Delta T_k)$ *:=<Operation><PositionX><PositionY><TeamID><PlayerID><BoardID>* (set of tokens separated by the white space separator).

For example if player 4 from team 1 placed a stone on board 3 at location 5,7 the preAction input variable would be "ADD 5 7 1 4 3"

$\vec{Y}(k\Delta T) := \{ServerAction, GameStatus, Teams, Allies, IllegalMoves, Winners\}$

ServerAction:=<Operation><PositionX><PositionY><TeamID><BoardID>

For example if team 1 placed a stone on board 5 at location 12,7, and the result of this was a capture of team 2 stone on board 5 at location 11,7 the *Server Action* message would be "ADD 12 7 1 5 REMOVE 11 7 2 5"

GameStatus:=<TimingMode><GameTimeLeft><TotalGameTime><StonesLeft><StonesAlive><BoardId><TimingRule><Time1><Time2><TeamTurn><StonesAlive><Prisoners><BoardState><ActionState>

TimingMode:=String Value *{"timing_sys_board", "timing_sys_team"},*

{GameTimeLeft,TotalGameTime,StonesLeft, BoardId,Time1,Time2,TeamTurn,StonesAlive, Prisoners}:= *Integer Value,*

TimingRule:=String Value *{"Pacing", "ForcedDelay", "TurnBased", "Independent"},*

BoardState:=String Value {"Pacing", "ForcedDelay", "TurnBased","Independent"},

ActionState:=String Value {“READY”, ”NOTYET”},

For example if the game is played on 2 boards with two different pacing schemas: Forced Delay and Pacing and timing measured against the board movements the *Game Status* message would be “timing_sys_board 28589 28800 49 50 1 ForcedDelay 19 0 1 1 0 running NOTYET 2 Pacing 0 0 0 0 0 0 running READY”

Teams:=<BoardID><TeamCount><List of Teams>.

For example if teams 1 and 3 are playing on board 1, and teams 2 and 3 are playing on board 2 the *Teams* message would be “1 2 1 3 2 2 2 3”

Allies:=<BoardID><AlliedCount><List of Teams>

For example if teams 1 and 3 are allies on board 1, and teams 2 and 3 are allies on board 2 the *Allies* message would be “1 2 1 3 2 2 2 3”

IllegalMoves:=<BoardID><PositionX><PositionY>

For example if a player had attempted moves which were illegal on board 3 at location 12,8 and on board 4 at location 6,7 the *Illegal Move* message would be “3 12 8 4 6 7”

Winners:= <BoardID><WinnersNames>

These are the entities which attributes are manipulated by the rule manager and game clients:

- Game
- Stone
- Board
- Position
- Team
- Player

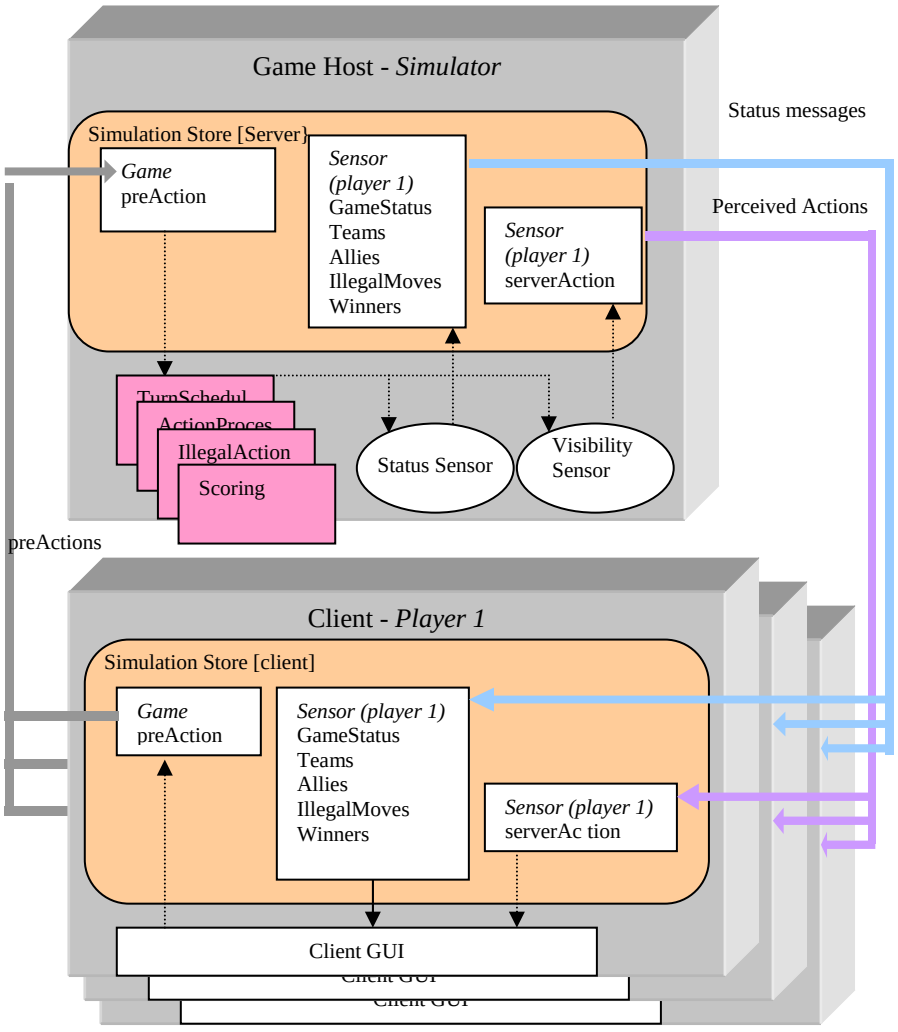

Figure 2. Go*Team Host/Client Model

The essence of the game play involves players placing stones onto the game board. The host may accept or reject stone placements. When a player attempts to place a stone onto their game board the client sets a preAction attribute for the Game entity in the local client repository store. As a result of the reflection mechanism it is automatically also set in the global simulation repository as depicted in Figure 2. During the deny phase, the simulator processes first the rule modules then the sensors. The Turn Schedule rule module processes the preAction attribute and accepts or rejects a proposed action. The Action Processor and

Illegal Actions rule modules validate appropriateness of that action and modify the state of other entity attributes in the simulation store as depicted in Figure 2 and Figure 3.

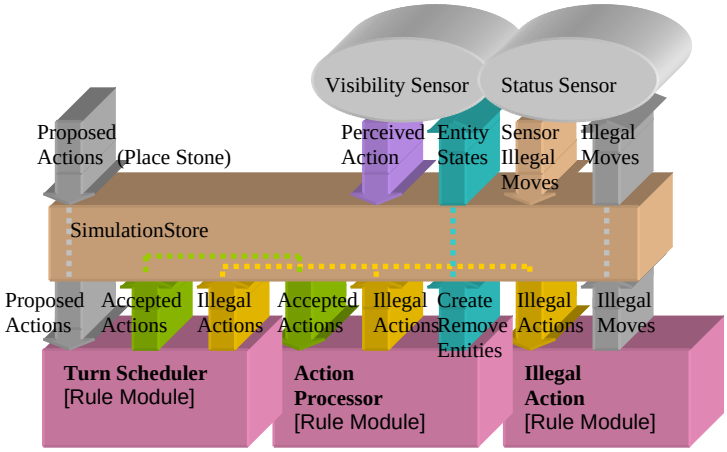

Figure 3. Processing Place Stone Request

3. RULE MODULES

The Go*Team game has implemented four fundamental rule modules: Turn Scheduler, Action Processor, Illegal Action, Scoring.

3.1. Turn Scheduler

The Turn Scheduler rule module is able to track different timing schemes for each board in the game. The following timing schemes for controlling the flow of the game have been implemented:

Independent Moves - Each team can add stones onto the board at any time they want. There is no restriction on the moves.

Pacing - Each team can only add one of their stones onto the board per time frame (relaxation period). The purpose of this is to slow the game down to give players an opportunity to plan the best move for their team.

Turn based - All teams are randomly ordered before the game starts. Every team has much time as they want to make their move, but they can only make a move when it is their turn. This blocking scheme can slow the game down considerably. Each board has their own order list of team turns.

Forced delay - This incorporates the turn-based theory where teams are randomly ordered at the start of the game but time plays a factor in this. After a stone from another team has been placed onto the board, the next team in order has to wait an amount of time (w) to observe the board, and then they are given an amount of time (m) to complete their move. If no move was attempted during the m amount of time, the team in play will miss their chance to add a stone onto the board. However, if the m amount of time is infinite, then blocking is introduced, leaning towards a turn-based approach with a delay. Forced delay is introduced to slow down the game, but it improves the result of the game where teams are forced to observe the game longer to prevent hasty moves, and at the same time are given a chance to attempt a move.

A different timing scheme can be specified for each board in the game. Alternatively the timing schemes can be specified on a team by team basis. The team based timing selection allows for the pacing scheme with a different relaxation period possible for each team. Giving teams a shorter relaxation period can provide a significant advantage as they are potentially able to make more moves.

3.2. Action Processor

The Action Processor rule module checks accepted actions to make sure that they comply with the game rules. Firstly, it must be verified that the position onto which the stone is being placed is available. Secondly there are some Go rules which may need to be checked against (if they are enabled) to prevent placement of stones which result in self capture (suicide, KO). If the action is legal, process actions will create an entity to represent that stone in the simulation store. Placement of the stone may result in the capture of prisoners from another team. The Actions Process rule module is responsible for working out which stones were captured and for updating the entity states in the simulation store accordingly.

3.3. Illegal Action

The Illegal Action rule module processes all the illegal actions and stores the illegal moves as player attributes in the simulation store. The status sensors will report illegal moves back to the players that attempted them.

3.4. Scoring

The Scoring rule module maintains and updates the scoring systems for each board in the game. There are a number of possible scoring systems and it is possible to configure different scoring systems for different boards in the same game. The

scoring rule module gets the score systems for each board from the simulation store as well as the team alliance information. The rule module uses the score systems for each board to update the territory count and prisoner count for each team in the simulation store.

When stones are surrounded/captured by allied teams, there needs to be a rule to define the owner of the prisoner stones. The Scoring rule module provides three prisoner ownership rules for the game host to use. The definitions of these rules are as follows:

Majority Prisoner Ownership Method:

Initial counting phase - each allied team is counted as one team. All other teams are independent. In this context, the winning team refers to the team that successfully captured the group/unit of dead stones.

Scenarios - if there is a tie between the allied and independent teams with regards to the number of stones used to capture the dead stones, the independent team takes priority, thus they get the captured stones.

If an allied team has the most stones used for capturing a group/unit of stones, the winning team of the alliance is the one with the most stones used to capture the stones. If a tie occurs, the last stone placed onto the board that caused the capture to occur, determines the winning team (using the stone's team id). If there's a tie between the allied teams (there can be more than one allied team), the winning allied team is the team with the last stone placed onto the board that caused the capture to occur. Then scenario 2 applies. If an independent team has the most stones used to cause a capture, they are the winning team.

Final phase - the number of captured stones (prisoners) is added to the winning team's prisoner count.

Proportional Prisoner Ownership Method:

Each team involved in the stone capture is given an even amount of divided prisoners.

Scenarios - If the number of teams involved can evenly divide the number of prisoners, then all teams involved will receive an evenly divided number of prisoners captured.

If scenario one applies but there are still some prisoners left (can't be evenly split across all teams), then scenario three applies.

If there are more teams involved than the number of prisoners acquired, then the prisoners will be distributed one by one to the most recent teams in order that caused the capture to occur.

Capturing Stone Prisoner Ownership Method:

The team that placed the last stone onto the board that caused a capture receives all of the prisoners captured.

The game host can change these prisoner ownership rules at any time during the game. In Go*Team the winning team is the team with the most territory points occupied on the board together with the number of prisoners captured. The overall winner is the player from the winning team with the most stones remaining on the board. When this has been determined (using the information from the simulation store), all players are notified of the winning teams and players on each game board.

4. GO*TEAM SENSORS

Each client has their own Status and Visibility sensor. These sensors are created by the clients when they join the game. They exist on the game host to report a perception of the current game state to the clients that created them. The sensors are executed after all rule modules are processed. The Status sensor reports any change to the game status, team configuration, alliance structure, inappropriate stone movements and the winners attribute while the Visibility sensors records any changes to the server action attribute in the global repository store. The visibility sensor has to work out which of the stones currently on the board are visible to which players. The algorithm for determining visibility could be just about anything. In the Go*Team case the following simple visibility rules are applied:

- from their own team stones, a player can only see their own stones. All other stones from the same team will not be visible to the player.
- from the opposition teams' stones, a player can only see those opposition stones which are closer (or same distance) to their own stones than any of their team mates stones. If another player on that player's team has a closer stone to the opposition, then the other player will see the stone

Alliances have no effect on visibility, allied teams are still treated as opposition teams for the purpose of visibility.

An implication of these visibility rules is that if only one player from a team has placed stones onto a board, that player will have complete visibility of every stone on that board. Conversely if a player has not placed any stones onto a board they will not see anything on that board.

The Game Host can see everything that is happening on all boards of a game. The game clients can only see the boards that their team is playing on. The player visibility sensor further limits what each player actually sees on their game boards. The Go*Team visibility rules can all be demonstrated using a simple example. Consider a situation in which 2 players from the same team are playing on the same board. White player 1 has placed one stone onto the board. White player 2 has not placed any stones on the board yet, and so they cannot see anything. White player 1 can see everything. The Game Host can see everything. The numbering on the stones at the Game Host shows the order in which the stones were placed onto the board (see Figure 4).

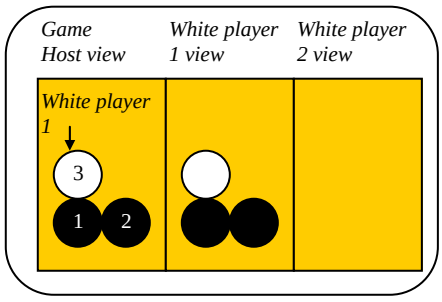

Figure 4. White Player 1 places a stone

Now if white player 2 places a stone onto the board, the visibility of black stones changes for both players as follows (see Figure 5).

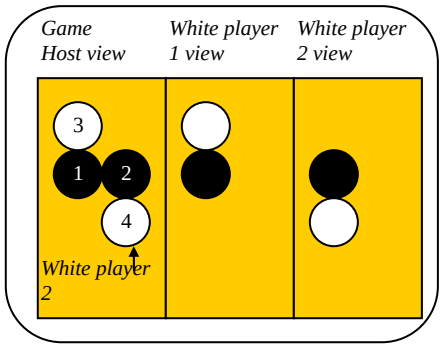

Figure 5. White Player 2 places a stone

When white player 2 places stone (4) onto the board, black stone (2) is no longer visible to white player 1 because someone on their team has a stone closer to stone(2) than they do. White player 2 can now see black stone (2) because they have a stone closer to it, but they are not able to see the other black stone (1). Note also that neither of the white players can see each other's stones.

5. GAME CONFIGURATION

The Game set up allows for configuration of the number of boards, game duration, board sizes, scheduling schemes, stone allocation and the number of teams alliances (see Figure 6).

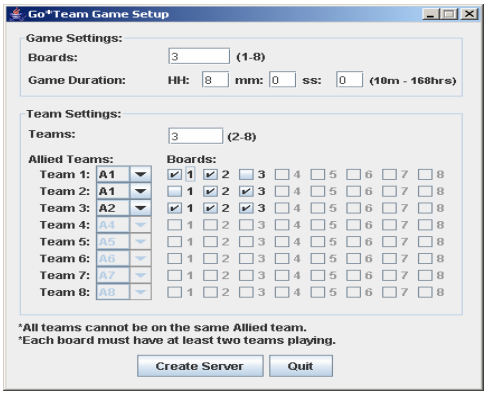

Figure 6. Game Setup

The host has a complete view of all boards and all stones, including the order of placement of stones (see Figure 7). On the game host all stones placed

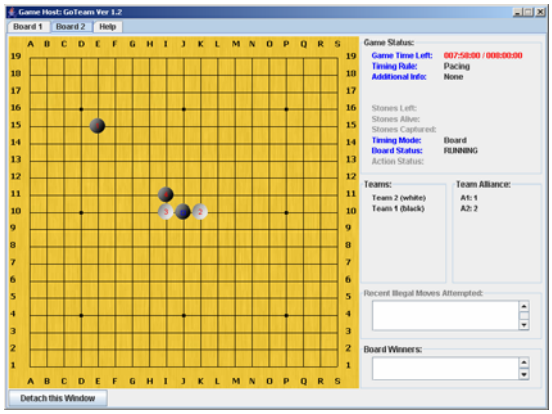

Figure 7. Host View of the Go*Team Game

by players appear on the game board. Each stone has a red number indicating when the stone was placed at that location. The most recent stone to be placed on the board has a blue square. The host does not show which stones were placed by which players.

6. CONCLUSION

The Go*Team game has been designed as a research vehicle for investigating collaboration and cooperation between team members in a competitive and dynamic environment. The Go*Team computerized game environment exhibits many of the features of a NCW

environment with its inherent uncertainties, ambiguities and complexities, information sharing, integration and overload issues, tempo, communication technologies, and the necessity of cooperation and coordination as well as the inevitable competition that seems to occur between different individuals and groups in such situations. The Simulation Framework was a natural feat for the implementation of the Go*Team requirements.

7. AKNOWLEDGMENTS

The authors are grateful to Nicholas Tay for his contribution into the design and development of the Go*Team game.

8. REFERENCES

Alberts, D.S., J.J Garstka, F.P. Stein (1999), Network Centric Warfare – 2nd Edition, *Sun Microsystems Federal Inc*.

http://en.wikipedia.org/wiki/Network-centric_warfare

http://www.britgo.org/intro/intro1.html,Introduction to the game of Go

Jagiello, J., M. Eronen (2007), A Simulation Framework, *MODSIM, Christchurch, NZ.*