Defence Science and Technology Group (DSTG) and Swordfish Computing Project

Distributed Decision-Making

THE UNIVERSITY of ADELAIDE

Final Report – a1734056

Group COMPLEX 8

a1734056	Hayden Lee

Project Vision Reflection

The project vision, as described in the initial report, was to create a user-friendly software system which can test different distributed decision-making (DDM) algorithms in a range of specified scenarios with a graphical interface. The final product managed to fulfil this vision and meet most of the expectations that were presented. These included having a modular implementation, capability to run scenarios with varying parameters to assess their effect on performance, and geo-spatial visualisation. Unfortunately due to time constraints, only one distributed decision-making (DDM) algorithm was implemented, so the software system's ability to compare effectiveness could not be demonstrated. However, appropriate performance metrics were still considered, and a method for comparing simulation outputs in a time-series display was developed. The project goal of producing modular, extensible, and scalable code was semi-fulfilled. The application frontend was built using JavaSwing, and features easy-to-use scenario generation tools. This is decoupled from the system backend, which takes advantage of the open-source Java Multi-Agent Simulation Of Neighbourhoods (MASON) framework to handle the simulation logic. This decoupling allows for future substitution of a different front/backend whilst still maintaining compatibility, and also facilitates easy integration of additional DDM behaviours once they have been developed.

Customer Q&A Reflection

Most of the team's Q&A was directed towards the product owner. During the sprint review meetings that they attended and through the team slack channel, the development team could clarify the project requirements. The key questions and answers have been presented in the Table 1:

Table 1 – Product Owner Q&A

Question	Answer
Is there a specific way you want to measure the outcome and an algorithm's effectiveness?	I'd like to see metrics directly from the algorithms, e.g., how many messages are sent, how many messages failed, etc. I would also like to see metrics from the simulation, e.g., which team won, how many times (if multiple runs), how many agents were neutralised per team, etc. Two levels of metrics – goal based on the scenario vs metrics on communication
What specific algorithms would you like us to investigate and compare?	It is up to you to choose the algorithms you wish to apply. However, your solution must be configurable such that many different algorithms could be applied in future. There are several different classes of DDM algorithms such as consensus algorithms. Be conscious of fault tolerance in algorithms; there's no point in comparing one algorithm with fault tolerance and one without if there are no faults in the scenario.
How do you want the inputs to the GUI to be represented? For example, number of cowboys, aliens, etc.	I do not mind how the data is input. It does not matter if the data is input via sliders, text boxes, or otherwise. The only thing that is critical is that the input is configurable. That is, the input may be changed.
What constitutes a complete user story? When is it acceptable to tick it off as an issue?	Look at the gaps in the user stories, we've done lots of work on all of them so identify and focus efforts on the gaps. None of the user stories are expecting a full integration test, just their intention.

	If you can do what the user wants, it's done – even if it's very simple and could be refined a lot. At the moment, all we need is a minimum viable product – don't need to extend beyond that.

In the final sprint review team meeting, the clients from Swordfish computing attended and were available for Q&A to give us some additional insights. The key takeaways from this session are presented in Table 2:

Table 2 – Swordfish (Client) Q&A

Question	Answer
We want to be able to run simulation multiple times and average results. How many (order of magnitude) times should it be run?	Depends how long the simulation takes to run but turning off interface sounds like a good idea. Largely depends on variance between simulations After doing 3-4, how much do they vary? If a lot, then we need to run more (100+) and if there is still variance, we have concerns and maybe we need to run 1000+
If running many simulations, do we omit front end GUI so it can be run quickly?	Good for investigation, a mode for running lots is good and there should be a mode for running less simulations but with much higher details – ie. Why did certain agents get selected? What other actions were available? Investigative mode vs variance mode. If we can see which messages are sent and choices are made, we want to be able to step through that. Maybe instead of a video feed, try a step wise feed – ie being able to stop and see what happens at each time (debug mode). Consider log levels in terms of messages, don't want to flood screen with unnecessary information, can filter based on log levels to show certain information and hide others .

Throughout the length of project development, these questions were the most important. The first three questions were about understanding the needs of the client, which played a significant role in shaping the direction of the project and guided some of the decisions that were made. One of the fundamental characteristics of the software system was that it had to be configurable. This meant that the software system needed to be relatively extensible so that additional elements can be implemented in the future with minimal integration issues. Another required aspect was the need for the algorithms and behaviours to produce results that could have their effectiveness compared. Since algorithms and scenarios that always produced perfect outputs would not be statistically significant, it was important to consider the need for failures to be producible through a mixture of parameter/algorithm selection. The recommendations for crossing off user stories was identified as a shortcoming for the group. Despite spending considerable resources on completing even one user story, the group found it collectively difficult to cross them off even if the product owner suggested that they were close to completion. This

may have lead to some stagnation as the development team would pursue a user story that already had a minimum viable product. The Swordfish Q&A also provided critical information about use cases from a research scientist perspective. Their insights into their rationale for increasing the number of simulations was not something that we initially considered. Their explanation for when the information displayed in the GUI would be good and when it wouldn't (head-more vs head-less) was useful in communicating specifically what they wanted to see, which paves the way for the project's future direction and what features should be implemented. Upon reflection, it may have been better for the development team to engage with these clients earlier to receive this advice, but prior to the meeting a lot of development effort was required to simply set up the framework for the software system. Receiving this feedback was essential to creating requirements for the final product, but addressing these and creating tasks from them would likely not have happened until later anyways.

Users and User Stories

The final user roles that the group identified were the research scientists from both Defence Science Technology Group (DSTG) and Swordfish Computing. It was decided to group the scientists from DSTG and Swordfish Computing into one category as 'research scientists' as both companies would be using the software in the same way.

The research scientists' responsibilities are to run the simulation and perform analytics on different desired configurations to compare how the algorithm operates under different configurations and parameters. An advanced potential action of the research scientists would be to implement another distributed algorithm, as the group only implemented the ring algorithm. If this was done, the algorithms' outputs could be compared against one another. The most important user stories the group implemented for the research scientists were; User Story number 7 – 'Easy configuration'. This user story read: "As a user with little technical experience, I want a simple, straight-forward way to configure the scenario, so that I can use the tool effectively without having to learn new skills." This was implemented through a Graphical User Interface (GUI) which can be seen below in Figure 1. As shown in Figure 1, the configuration of the GUI is simple; cowboy and alien agents can be added or removed, and their parameters can be altered. Their locations are specified by entering X and Y coordinates and their attributes, such as the aliens' movement speed, are set with sliders. This can all be achieved without having any technical knowledge or learning any new skills.

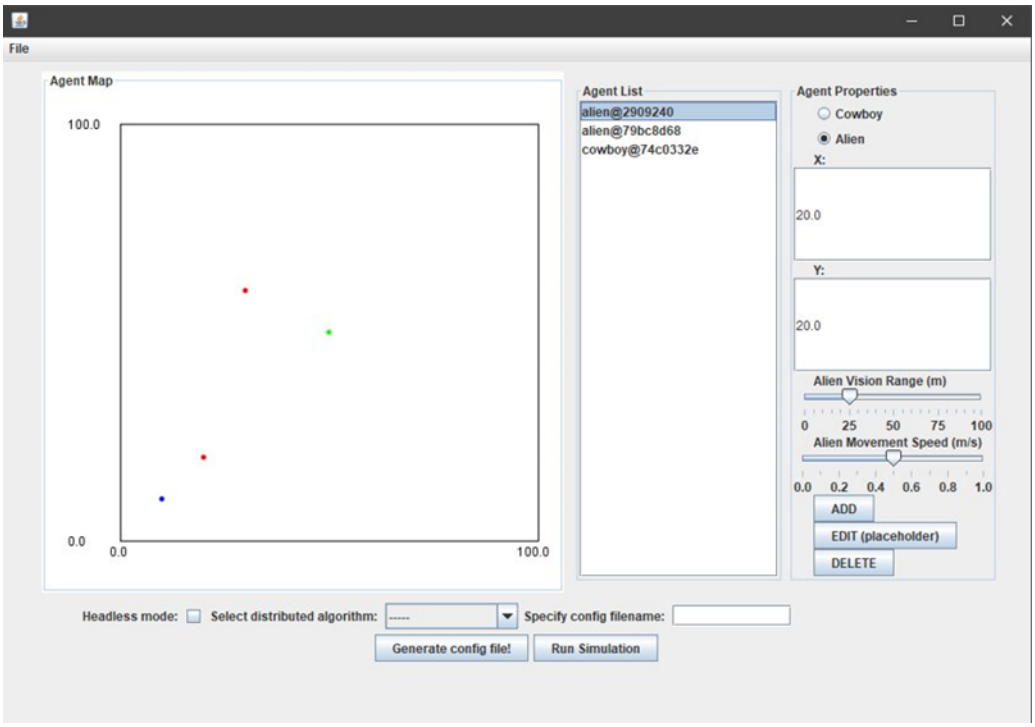

Figure 1 – Simulation Graphical User Interface

Through the graphical user interface shown above, user story number 3 was also implemented. User story 3, 'Configuration Profiles', reads: "As a recurring user, I want to store configuration settings, so that I don't have to spend time re-entering parameters". A user can create a desired setup of the simulation and then click "generate config file" and a JSON file is created which can be imported into the GUI and re-used as the user desires. The config filename can be specified, so a library of many different configurations can be developed over time and loaded with ease. Another key user story implemented was User Story 1 – 'Algorithms and Choices', "As a typical user, I want to be able to choose at least one algorithm to test, so that I can see how effectively it performs". This user story was achieved as the ring algorithm was successfully implemented. The front end also has extensibility to be able to select from multiple algorithms if the research scientists desire to implement other distributed algorithms.

The final key user story was User Story 5 – 'Time Series Display', "As a user interested in how the algorithms perform over time, I want a display that can show time series data from the simulation, so that I can see how the scenario evolves over its duration.". At the completion of the simulation a CSV file was created which contained all relevant analytical data. This CSV was then called upon and the data was visualised. The user story specifies the display of time series data, this was achieved through a visual which graphed the number of messages sent at each time step of the simulation. This is shown below in Figure 2.

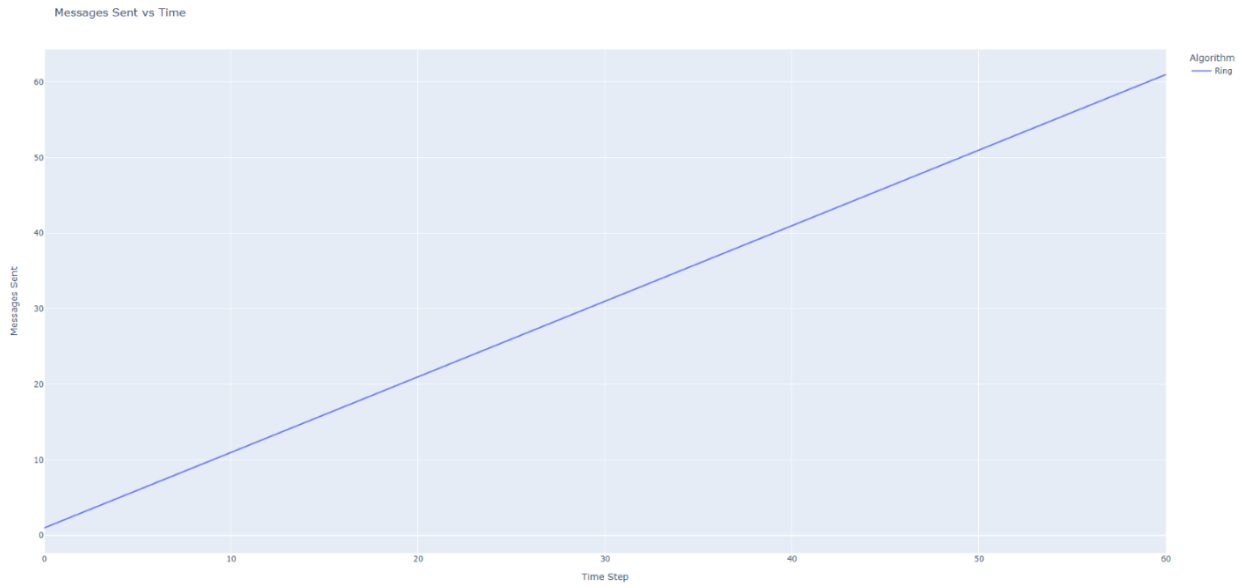

Figure 2 – Time Series Display of Messages Sent

The roles imagined in the initial report read: “The end user of the software package will be the DSTG scientists. Ultimately, the scientists desire a system that allows them to compare the effectiveness of various distributed making algorithms. Using the software would consist of selection of the simulation parameters, selection of the DDM algorithm of interest then running the simulation. Following this, the scientists would then expect to obtain the results of the simulation.” This is very similar to the roles of the research scientists at the conclusion of the project. The only difference being that the software developed only has the ring algorithm implemented, thus the research scientists can’t currently select or compare the effectiveness of various distributed making algorithms. However, they can select simulation parameters and DDM algorithms through the GUI, save configurations, simulate a scenario with the ring algorithm and view the results of the simulation. This satisfies the majority of the user stories and identified use cases.

Software Architecture

The software architecture diagram in Figure 3 shows the organisational structure of the software system frontend/backend and the major interfaces connecting them.

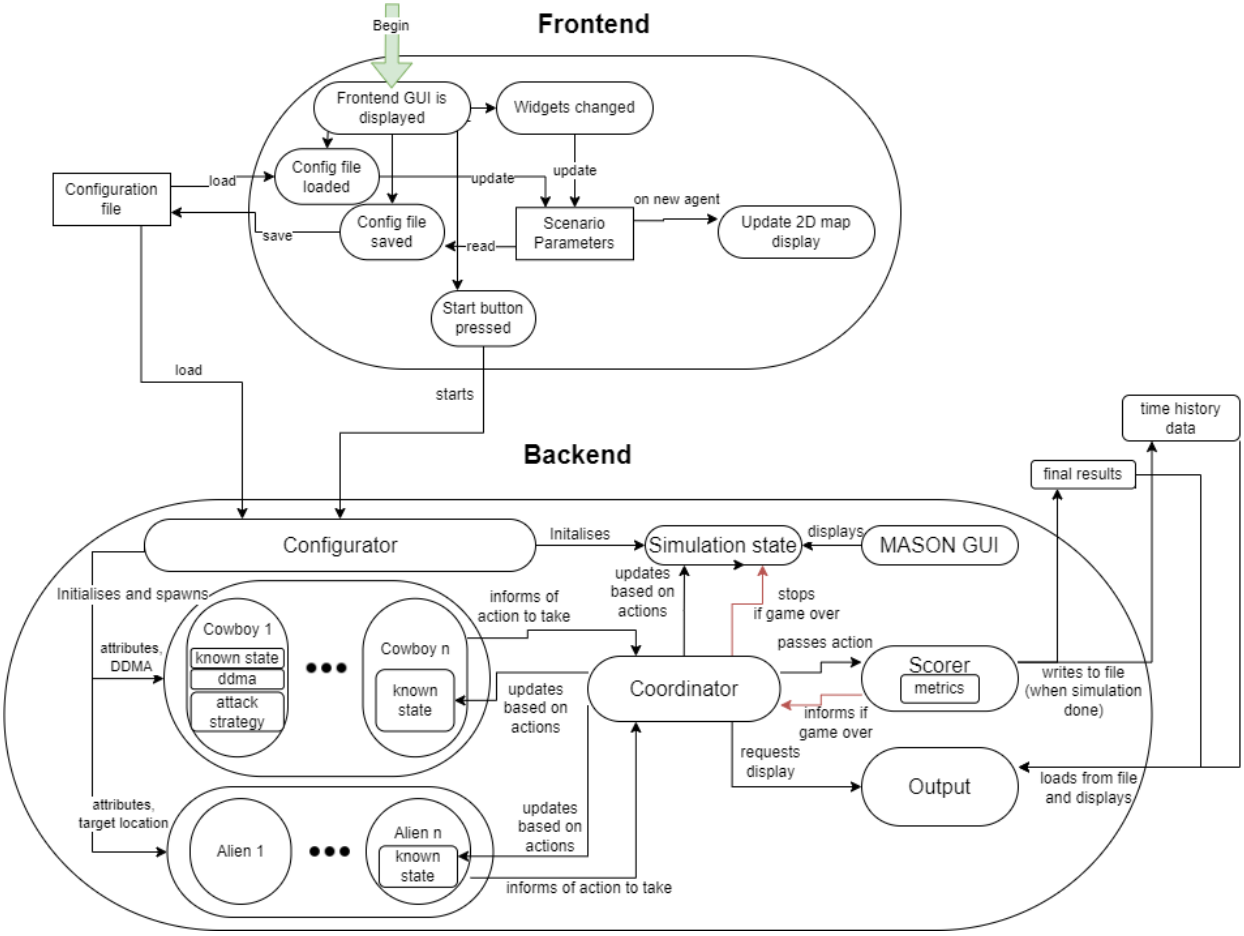

Figure 3 – Architecture Diagram

The project architecture consisted of a layered organisational style, an object-oriented decomposition style, and an event-driven (broadcast) model. The software system naturally fit a layered organisational style as the design principles needed to revolve around the agent-based simulation framework that the development team would decide on. If the simulation framework acted as the system's business logic layer, it would require support from a presentation layer consisting of a user interface, and a data access layer that contained all the underlying information required to run the simulation. To facilitate a simulation of agents with varying parameters, it also made sense to take advantage of an object-oriented decomposition style. By creating each agent as an object and maintaining a record of object lists, the models can be implemented across all layers of the software system with minimal complexity. Lastly, the broadcast model was utilised as the software system control style. This is entirely controlled by the coordinator, which acts as the single mechanism for manipulating the SimState as it is always aware of the entire software system state. The control flow between subsystems is entirely managed by system events. This primarily included clicking on buttons, selecting agents on the user interface, and

simulation events completing. An event-based control style was easy to implement; however, it will be important to ensure that future subsystems are managed by the coordinator appropriately.

Tech Stack and Standards

The final project tech stack involved a decoupled frontend configuration file generator GUI and a backend scenario logic simulation engine. The frontend was written using JavaSwing, a GUI widget toolkit for Java, that provides a set of GUI components used to generate user-created scenarios with corresponding configuration files to read into the backend. The framework for this consisted of a main window containing panels used for displaying scenario information and interactable elements for manipulating agent creation. When called, the backend will start performing the simulation using the frontend-generated scenario. This is handled by MASON, an open source multi-agent simulation environment configured in Java. Upon completion, the simulation logs are passed to an external python program to produce a time-series display of key performance metrics to assess algorithm effectiveness.

The team used Git as a distributed version control tool, which was hosted on a GitHub repository, which allowed each subteam to work separately on different branches. It allowed for incremental software changes to be tracked, such that features could be added without causing accidental interferences. Using GitHub as a host also allowed integration into GitHub projects, which was used to organise the team's sprint activities, user stories, and weekly tasks on a product backlog board. Slack was used as a platform for team communication. A main team channel was used for project discussion and communication with the product owner, whilst secondary subteam channels were also created for more targeted discussions. A Slack bot called Dixi was also used to automate daily standups. The team used VSCode as an IDE, which had suitable debugging and refactoring tools to aid with software development, as well as featuring Git integration. Whilst this was adequate for fixing bugs and catching things such as unused parameters and methods, we would have liked to use tools for specifically performing static and dynamic code analysis. We ultimately did not explore options for these, as a lot of the development effort was used to focus on learning about the specific front and back/end architecture that we wanted to use for our software system. Although these tools could have helped find instances of inefficient code, complex expressions, and other general bad practices, producing a working demonstration was the top priority for the team, and this could be accomplished using the built-in tools offered in VSCode already. VSCode was set up to autoformat to the Google Java coding standard.

The final tech stack worked well and caused minimal issues in the final product. The tech stack in the initial report featured AnyLogic as the most likely choice for an open source agent-based simulation environment; however, it was found to be unsuitable for collaboration. Instead MASON was chosen to be used for the software system backend. Although it was difficult to set up at the start, it featured dynamic visualization, agent communication, data logging capabilities, and configurability. Since AnyLogic supported Java as a programming language, JavaSwing was the first choice for the frontend. This was maintained throughout the project as MASON also facilitated Java integration and so this justified the team's decision to stick with Java.

Group Meetings and Team Member Roles

Our project development team met weekly to discuss progress with sprint tasks, with subteams meeting informally where needed. The regular weekly meetings were successfully timeboxed between 11 am to 12 pm, which was sufficient in ensuring that the team had enough time to discuss all items on the

agenda. This was possible as a meeting agenda was prepared and distributed by the scrum master each week which everybody could add to. Whilst staying within the timebox was difficult in the first sprint, we were able to reflect on why this was the case in the first retrospective and understood that this was due to too much time being spent on status updates. To remedy this, the team committed to using the standup bot with more discipline and to write a short paragraph in a summary document at the end of each week, instead of running through a progress update during the meeting. Every second week, these meetings were also scheduled with the product owner for our sprint reviews. The sprint retrospective team meetings were generally scheduled on the Friday before the upcoming sprint review and were timeboxed for 30 minutes. Apart from two group meetings that the customer (Swordfish) attended, no additional communication channels were opened with them as all of our queries and feedback could be directed to the product owner, which facilitates the "fail fast, fail often" agile methodology.

The Scrum Masters for each sprint are described in the Table 3.

Table 3 – Sprints and Scrum Masters

Sprint Number	Weeks	Scrum Master
1	2 - 4	Hayden Lee
2	5 - 6	Nathan Van der Hoek
3	7 – 8 (incl. mid-semester break)	Vinh Nguyen
4	9 - 10	Patrick Capaldo
5	11 - 12	Harry Bagley

Group meetings were always very organised and productive. Each time, all team members had something to discuss and were happy to contribute to the conversation. The scrum masters did a great job at maintaining momentum throughout the meetings and at ensuring that discussions were focused. The quality of the meetings definitely improved throughout the semester. At the start, the development team was still fresh to the project and so it was difficult to find tasks to work on in the first sprint. During this time, there was a sense of uncertainty with what to pursue, so this also made the meetings difficult. However, the project finally started to take shape in the third sprint and the team had great initiative with breaking down user stories into sprint tasks which significantly improved workflow and the meeting quality. This also facilitated the natural development of subteams as group members started developing fields of expertise. These included:

- GUI
- Algorithms
- Scenario/Simulation
- Metrics/Output
- Integration

It should be noted that team members were not permanently assigned to a single subteam. As the sprints progressed and the focus shifted between user stories, it was natural for subteams to change as well. The changing requirements of each sprint was easily picked up by the entire team which encapsulated the agile work process methodology. With subteams working and collaborating separately, the meetings were a chance to share their incremental progress and receive feedback from the rest of

the team. This was a very successful approach and speaks to great length about our agile project management style.

My primary focus for the project was the frontend GUI, although I also spent time assisting with metrics/output and the early stages of scenario/simulation. I also lead the team as the scrum master for the first three week sprint. During this time, I tried hard in leading the team meetings and organising sprint activities, but I found it very challenging. Starting the project from scratch meant there was a lot of uncertainty with what areas we should focus on and explore, although I can be proud to say that I managed to organise the team into our first subteams where we could separate and start working individually. From this we were able to generate our first concepts and demonstrations, but I was very relieved when my scrum master duties were eventually over. Afterwards, I was able to dedicate my efforts towards learning JavaSwing and creating a user-friendly configuration file generation application. I worked closely with my subteam partner to create an initial version that contained all the fundamental functionalities of the software system. After the mid-semester break this was completely reworked to make it better fulfil the requirements of the user story. It was very rewarding to work on this part of the project as I could appreciate the incremental improvements that I was making with each iteration. Seeing the entire software system come together with the frontend and backend integration was very pleasing to see.

Snapshots

See attached zip folder.