

# **“Agents with Attitude” – Adapting teamwork models to add narrative depth to squad strategy games**

**University of St Andrews**

**Submitted 22/01/2021 by Victoria Goad**

**Supervised by Alice Toniolo and Christopher Stone**

## **Declaration**

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 7,345 words long, including project specification and plan and not including the appendix.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

## Table of Contents

Declaration.....	1
Abstract.....	2
Introduction.....	2
Objectives:.....	3
Primary:.....	3
Secondary:.....	3
Context survey.....	4
Agent and Teamwork modelling.....	4
Natural Language Generation.....	6
Ethics.....	7
Design and Implementation.....	8
General development process.....	8
Game.....	8
Teamwork model & argumentation.....	12
Natural Language Generation.....	14
Testing and Evaluation.....	15
Personal testing and Example Output.....	15
Conclusion.....	17
Appendix.....	19
A: Bibliography.....	19
B: Initial Time Plan.....	20

## Abstract

This project uses Java to implement a turn based strategy game which is played by teams of autonomous agents. These agents use numerical heuristics based on logical checks to rank the perceived value of their available moves and communicate their most valuable move to the rest of the team. The team then proceeds to hold a discussion to select the most valuable move and communicates this to the core game loop in order to carry it out. This discussion is rendered into English via natural language generation and presented to the user as dialogue between the characters represented by each unit. The aim of this project is to have this game present a more immersive narrative experience than other games of the same genre by presenting the in-game units as convincingly human-like actors. This particular implementation serves as a basic proof of concept for these aims, to which more refined agent and communication systems could readily be added.

## Introduction

There exists a somewhat broad and ill-defined genre of video games which, for the sake of clarity in this report, I will refer to as “squad strategy” games. Examples of this genre include the XCOM series, the Wasteland series, Darkest Dungeon and numerous others. I believe that the main trait that unites these games into a single genre, regardless of their often very different narrative themes and gameplay elements, is their focus on controlling a small team of units which are unique, or at least

distinct from one another, in terms of their abilities, and very frequently in terms of their looks and personality as well.

In some examples of the genre, such as XCOM, these units' identifying features (including their physical appearance, armour, and accent or language used for voiced lines) are procedurally generated from a pool of individual components, while in some other games like Darkest Dungeon, a unit's class defines much of their appearance and dialogue, though their names and some personality traits can still be randomised. In most games of the genre, elements of a unit's appearance not closely tied to gameplay can be quite readily customised as well. In my own anecdotal experience, it is a very common trend for players of and fan communities for these games to become highly invested in these units, either as their procedurally generated selves or when they are edited to represent, for example, friends of the player, members of an online content creator's audience, etc. In such cases, they frequently become anthropomorphised; they cease to merely be *units* and instead become *characters*. Players will assign personalities to them, celebrate their victories, mourn their deaths and frequently share their stories with other fans. In short, I believe this phenomenon contributes to the popularity of and enjoyment provided by these games, in a significant and unique way.

My project intends to capitalise on and improve this aspect of the squad strategy game by creating autonomous units that are intended to be as convincingly human-like as I am able to make them, ideally creating an experience in which this anthropomorphism and emotional investment can be more easily achieved and wherein players experience a greater degree of investment. In order to accomplish this, I have identified three main aspects to focus on implementing into this project:

- A squad strategy game of sufficient depth and complexity to facilitate a variety of game states and strategies for the units participating in it. This game will also provide context for the agents' interactions, and ideally some input into their simulated mental or emotional state.
- A set of autonomous agents, each capable of controlling an individual in-game unit. This will entail them being able to identify both what actions their assigned unit can take within the current game state, and what actions are *preferable* for them to take. Additionally, these agents should communicate their planned actions to other units, and the team as a whole should discuss what action or set of actions is optimal for the team as a whole.
- A system of natural language generation which converts the messages between agents into English and presents them to the user, displaying them not as part of a communication protocol, but as dialogue between human-like characters; specifically, dialogue which is convincing and appropriate to the unit's assigned personality.

## Objectives:

### Primary:

1. Design and implement a squad focused, turn-based strategy game of sufficient complexity to allow for a wide variety of potential actions and plans.

2. Implement an agent model capable of making proactive decisions and plans on the level of individual units in the game, establishing plans via argumentation and coordinating these plans across a team of similar agents.
3. Extend the agent model to include a system of personality traits/values, which differ between agents and substantially affect their decision making process.
4. Implement a means to translate communication between agents into English and present it to the user in the form of dialogue between in-game units/characters.

## **Secondary:**

1. Extend the game to include a graphics system which allows users to readily understand what actions are being taken
2. In addition to the above objective, implement a “mood”/emotional state system which can also influence decision making and changes in response to observed events in the game.
3. Expand the agent interaction system to allow for more “organic”/“human-like” behaviour such as dissenting action based on disagreements in the planning layer, or an internal model of other agents which encodes opinions of or personal relationships with their units.
4. Expand the English translation system to produce a variety of translations for the same arguments and symbols, potentially also affected by personality and/or mood.

Of these objectives, I believe that all of the primary objectives have been mostly fully implemented in this project submission, with some exception for objective 2, as the provided agents do not have the ability to plan or co-ordinate over the course of multiple actions. Otherwise, the project features a turn-based strategy game which allows for a broad agent decision space, agents which evaluate and communicate about these decisions, a basic personality system and a system which converts their communications about said decisions into pre-written English sentences.

Some secondary objectives are partially fulfilled as well. Objective one is fulfilled by way of ASCII output to the console which, although basic, does fully demonstrate the current state of the map and the positions of all units on it. Likewise, the English translation system features multiple voices and multiple possible sentence options for each individual token. Dynamic emotional states, and more complex agent interactions such as opinions of other agents or dissenting actions, could not be implemented within the time-frame of the project.

## **Context survey**

I was unable to find literature specifically concerning the games which inspired my approach to this project’s game design component, or at least ones directly relevant to the aims of the project itself. For this reason, I relied mainly on personal, anecdotal experience or non-academic research to inform my game design decisions, and focused my context survey on the remaining two aspects of the project.

## Agent and Teamwork modelling

Models for autonomous agents capable of teamwork have a long-standing history in scientific literature. Established and relevant historical uses for such models include applications in simulated military helicopter operations<sup>[1]</sup>, which presents useful points on the topic of coordination with potentially limited communication and dynamic role assignment, and in robotic soccer matches<sup>[2]</sup>. The prevailing model architectures are the Soar and ACT\* families, with Soar more focused on achieving functional and rational behaviour, while ACT\* attempts to more closely replicate the details of human behaviour (clarify with further reading; emotional state, etc.)<sup>[3]</sup> This would suggest that an ACT\* model would be more likely to fulfil emotionally driven, potentially “irrational” or convincingly human behaviour, and therefore is more likely to fulfil the secondary objectives of this project.

Especially in recent papers, these models have increasingly overlapped with or borrowed elements from psychological theories of mind, especially in situations where it is important for them to collaborate or otherwise interact with humans.<sup>[4]</sup> In one research paper, focused on creating a convincing agent for personality and emotion simulation<sup>[5]</sup>, makes use of psychological models for specifying and codifying personality traits (examples given in the paper include the OCEAN model, which tracks features such as openness, conscientiousness, extroversion, etc. along with their opposites). This serves as a solid basis for decision-making factors besides raw utility, or perhaps more accurately, it provides different means of evaluating said utility. For example, a personality model suitable for this project could feature an aggression/defensiveness dimension, wherein certain agents may place more value on killing enemy units, while others may place more value on protecting allied units.

Communication for a model of this type will require a co-ordination model in which different beliefs and goals can be communicated and subsequently evaluated, so that a main strategy for the team as a whole can be established. One option presented in research literature is an argumentation model<sup>[6]</sup>, wherein internal knowledge and subsequent decisions are encoded into formal logic arguments, and subsequently presented alongside each other in a dialogue game-based protocol. This protocol attempts to ensure that when an agent presents a course of action, all arguments relevant to that action (and any subsequent responses ) are presented and properly connected to each other. This paper also proposes an abstract method of resolving these connected arguments into a directed graph, and reducing this graph such that arguments providing differing justifications for the same point can be combined and arguments which are weaker than their counter-arguments can be removed, eventually resolving to a single or a small subset of agreeable actions.

Papers on team-specific decision making models have also raised the prospect that ideal teams, especially ideal *persistent* teams, should be able to reason about not only their own state or the general state of their environment, but also specifically identify and reason about their team, in the present and future sense.<sup>[7]</sup> This, in other words, allows team agents to reflect about their own role in the team, and the expected long-term utility that it would result in. This idea of long term persistent planning and resource management is extremely important to decision making in the context of games such as XCOM; one example given in the cited paper about evaluating the long-term cost of using a helicopter in the simulated military operation scenario maps very cleanly onto the cost of losing a unit, both in the context of an single encounter (in that losing a unit makes the

overall team less effective for the remainder of that encounter) and in a persistent environment (in that replacing units often represents a significant expense in terms of time and resources).

The agent decision-making process under models such as STEAM is heuristic, and while more formal problem solving systems exist, they encounter increasing complexity, especially in situations where individual agents have an incomplete view of their whole environment.<sup>[8]</sup> This form of decision-making presents an alternative underlying problem solving system should the baseline system of a more established teamwork model prove to be unreliable, by instead relying on efficient, structure-based solving of Markov decision spaces.

This project operates in a somewhat difficult problem space for team role allocation. While a pre-defined, hierarchical role structure can be applied to this context, it is at odds with both the argumentation model of group decision making and the possibility that agents will have their assigned units die (and, for the sake of verisimilitude, should therefore no longer be able to contribute to team discussions). Furthermore, a team will need to create, carry out and switch between multiple plans over the course of a game. It would therefore be more suitable to make use of a dynamic role allocation system, algorithms for which have been laid out in papers such as this one.<sup>[9]</sup> In this algorithm, role assignment problems are formulated as a generalised assignment problem (GAP)<sup>[10]</sup>, which formalises a team's available agents and the capabilities of those agents, and describes the problem and the optimality of its solution as a numerical function of the roles that need to be assigned in order to fulfil that task (essentially, its sub-tasks) and the configuration of agents which can be assigned to those roles. Optimal role allocations are ones in which the capabilities of the agents involved are maximised, while more advanced versions of the algorithm (also presented in Evolution of a Teamwork Model) optimise this allocation over each time step of a long-term plan, and later do so in an approximate, response time-conscious fashion. While version of the algorithm is more relevant for an environment which requires real-time decision making, its principles may be worth considering if turns begin to take excessively long, or to give the illusion of real-time decision making on the part of the units themselves.

Heterogeneous teams, or teams that involve both human operators and AI agents, occupy a distinct research space, which can be connected to this project through the potential involvement of human direction over the in-game agent team. Some earlier papers in this field involve agents which model a partner they may not be able to directly communicate with, and adjust their behaviour to compensate.<sup>[11]</sup> However, in the context of this project, it may be more convenient to look to "AI advisor" implementations of heterogeneous teams.<sup>[12]</sup> This paper, for example, raises potential team utility issues, in that friction between human and AI model decisions, or human operators lacking confidence in the AI model, may lead to overall reduced utility. (This is, however, confined to a context wherein optimal problem solving is the main goal of the project. In this project, by contrast, interesting narrative experiences are the main focus, and in this case friction between human and agent decision making may lead to more interesting events and convincingly human agents, assuming it does not unduly frustrate the user.)

The teams for this project will also be heterogeneous in another sense, in that different agents will have access to different actions, based on the abilities available to their assigned unit. One solution proposed in this paper<sup>[13]</sup> is the use of modules. In such a system, the logical rules required to reason about and carry out certain actions are written into a self-contained module specifically connected

to that role (say, a particular unit type), and an agent is created by combining a number of modules. Under this framework, agents also broadcast what modules they are comprised of, and therefore what actions they are capable of, allowing plans which require specific capabilities to be formed and for the appropriate agents to be assigned to contribute to those plans. This would also allow for more flexible development in the style of an object oriented language, as shared rules for communication, environment navigation or common actions could also be encoded as modules which are simply used by all agents, regardless of unit type.

## Natural Language Generation

This paper<sup>[14]</sup> provides a summary of the theoretical definitions of and some approaches to natural language generation. NLG is, essentially, the process of converting some form of input data, frequently one which is difficult for humans to interpret, into natural language; in the paper, this is accomplished through several steps of varying importance based on the type of input and intended form and complexity of the output. For example, component tasks such as content determination task may require less focus due to the nature of the input data as a team communication protocol, which is already formalised and should not generate much spurious or disjointed data. Likewise, text structuring and aggregation should be made simpler by the fact that the input data already possesses a certain amount of grammatical structure due to this protocol, though it may be useful to include these tasks to some degree, possibly by combining multiple responses from a single agent to a given argument whenever those responses make the same point with different justifications. This paper also discusses generalisable data driven approaches to NLG architecture, which may be too labour intensive for the scope of this project, though some aspects presented alongside those approaches may be useful, such as the ability to randomly select synonyms for the sake of more naturally varied, less formulaic output. One such data-driven model, as discussed in this paper<sup>[15]</sup>, makes use of reinforcement learning. Though still potentially beyond the scope of this project, the paper outlines several advantages over traditional NLG models; namely, the ability to more readily adapt to changes in context, to generalise to new inputs and the potential for data-driven development, which would obviate the need to manually make additions to the lexicon.)

One development in the field which seems particularly relevant to a project such as this is the concept of a realization engine, which implements a portable system for constructing grammatically sensible, linearised sentences using a data-to-language mapping which can be supplied to it by developers.<sup>[16]</sup> This allows developers to focus specifically on word choices and translations relevant to their input data without also needing to develop a grammatical model of their own, allowing us in this project to specifically tailor the NLG component's lexicon to game events and terminology, and allowing more time to introduce variance into this lexicon. This paper<sup>[17]</sup> goes over the significant effect word choice can have on the clarity and quality of user engagement with an NLG-generated text, and additionally demonstrates the potential of analysing human-produced texts in the intended field for their word choices, and refining this pool of available words based on the frequency with which certain words are used, preferences between synonyms, etc. A similar process could be applied in this case to create a lexicon which more closely matches real-world communication (either in video game or military contexts) and which more clearly and accurately represents concepts in the argumentation layer.

The specific intersection between argumentation systems and NLG is not novel; This paper <sup>[18]</sup>, for example, presents an argumentation framework wherein arguments can be qualitatively compared by the combined quantity of decisions which comprise that argument and the number of redundant points raised by that argument (amounting to a series of boolean logical rules), and an NLG algorithm which checks for and exposes those comparison conditions based on a decision tree structure. Based on the degree to which this project's argumentation model relies on binary ordering via logical rules, this paper may inform a similar approach to the NLG implementation in this paper, though some care should be taken with such an approach as strictly formal NL arguments may seem contextually out of place within the narrative of the game in a way which is not an issue for a legal environment. A similar problem space and corresponding planner is described, albeit at a somewhat abstract level, in this paper <sup>[19]</sup>, which describes a decision making system based on formal logic and if/then rules, which are then expressed to the user either in the form of logical symbols or natural language expansions of those symbols. It also discusses the implementation of user-prompted explanations, and how the system might behave at various levels of autonomy and/or assumed user approval for any given action, which may be useful considerations should this project be developed far enough to allow for human input and heterogeneous team structures.

## **Ethics**

No ethical considerations apply to this project. Originally, I intended to conduct an artefact evaluation by providing my peers with the game and a limited and anonymised feedback survey, but I did not have a minimum viable product before the end of the semester and hence was unable to conduct an evaluation in this way. A copy of the ethics self-certification form is included alongside this report.

## **Design and Implementation**

### **General development process**

Development for this program took place in an iterative fashion based roughly on the Agile model of software development. During my weekly meetings with my supervisors, we would discuss progress made on the project in the last week, newly added features, and objectives to focus on for the next week, usually by describing desired functionality in a specific but still higher level form- for example, we might specify in a meeting that the agent model should include a "basic personality system" by next week, but I would carry out the act of breaking the implementation of that system down into smaller and more specific tasks independently and informally.

Bug testing and fixing was carried out on a similar iterative, informal system. I would conduct test runs of new features focusing on core behaviour and potentially prominent edge cases, identify critical issues (usually in the form of program errors or unexpected behaviour) and resolve them before moving on to implementing new features. I had intended to conduct a more thorough quality assurance pass once I had implemented the majority of the project's intended features, but this was not possible, as implementing these features to a satisfactory degree took longer than anticipated.



# Game

While I initially considered attaching my agent system to an existing squad strategy game, such as one of the instalments of the XCOM series, this ultimately proved to be infeasible; Even OpenXCOM, which has the most robust scripting component of any suitable game I could find, did not make enough of the game state visible in code form for me to create a suitable agent input layer. The only alternative would be having the agent parse the game's graphical output, which would have been far too complex for me to implement on my own during the time-frame of this project. OpenXCOM had the additional issue of utilising a mechanic known as "time units" (Tus), wherein units do not have a fixed number of actions, but instead a pool of points which they can expend to carry out many combinations of actions in one turn, consequently massively expanding the decision space available to any agent I would construct.

Hence, I opted to use Java to implement a simple game based heavily on XCOM: 2, the sequel to the game's 2012 reboot. Coding my own version of the game from scratch would make it trivial to access any parts of the game state required for agent input or decision making, while the newer XCOM games make use of a vastly simplified action system wherein units have one move action and one non-move action (which can also be used to conduct an extra move) per turn, making it much simpler to track and identify what actions an agent can or should use.

Additionally, the newer games feature a class system in which different types of units have access to different abilities, not tied solely to the equipment they can be given on a strategic level. For example, XCOM 2 features a set of four main classes that includes a close range melee/shotgun user, a sniper, an explosives expert and a support class which can use a remote controlled drone to heal and support allies or conduct electronic warfare. This ensures that each unit has a more diverse set of actions and therefore a more distinct decision making process, and opens up space for combinations of supporting actions and the evaluation thereof in the agent planning and communication phase. Therefore, I opted to create approximations of these classes in my own game. In this case, this comprises a medic class capable of healing other allied units and applying an effect to them that makes them temporarily harder to hit, a demolitionist capable of destroying cover (which turns it into an empty space) and throwing a grenade which can deal guaranteed damage in a small area, a sniper with an especially long-ranged and accurate special attack and the ability to hide (which provides a similar defensive buff to the medic's, but only to the sniper themselves), and an assault class which can use a melee attack that requires them to move adjacent to enemy units, but which also deals a large amount of guaranteed damage.

As the main purpose of this game is to serve as a test bed for the agent system, and to provide input to and context for it, I deliberately chose to implement only those aspects of the XCOM games which would serve to highlight the agent system in action while minimising other aspects. For example, my implementation does not include the "Geoscape", a strategic layer which takes place in-between combat missions and focuses on resource and base management, as the units which the agents control do not take their own actions in layer. Similarly, my game relies exclusively on the Java terminal to output dialogue, along with the game state as rendered in ASCII form.

While XCOM does feature unit persistence between combat missions, with units obtaining new abilities and equipment over the course of multiple encounters (assuming they survive), I determined this to be beyond the scope of this project, though I believe it would be a valid extension to include a system which can induce changes in an agent's personality or mindset between missions as they learn from or are "emotionally" affected by their experiences.

As it stands, the game exists in the minimal state required to enable and demonstrate the agent system. In this state, it runs autonomously without the need for, or indeed the possibility for, user input, generates simple random maps and has a base victory condition wherein one team wins when the other is completely eliminated. The control flow of the main game process is outlined in the diagram below.



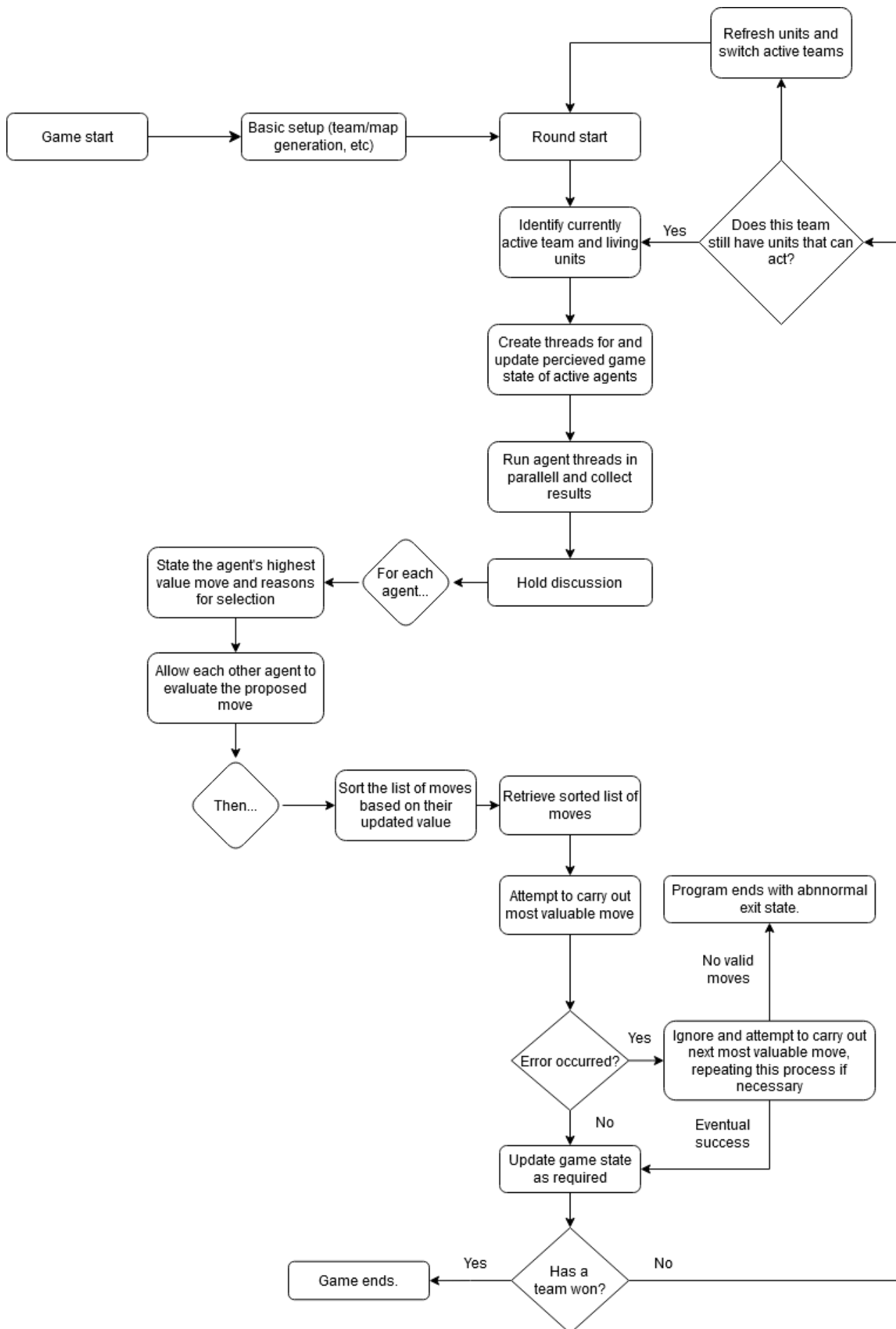


Figure 1: A flow chart describing the game's main method.

The classes and objects used to facilitate the game itself are based on relatively simple and efficient object oriented Java constructs. The map itself is a two-dimensional array of tile objects which store the contents of a given co-ordinate as attributes, allowing for memory efficient storage and quick coordinate based lookup and taking advantage of the fact that the map's dimensions, in the current implementation, form a square, the size of which remains constant over the course of the game.

Units take advantage of Java inheritance; a central "entity" class contains attributes and methods common to all units in the game in a Protected form, including base statistics such as health and movement speed, and actions shared across all units, such as basic movement and attack actions. This includes helper function used to access an A\* algorithm, the implementation of which was sourced from an open-source GitHub library which is credited in the source code for the A\* classes. Any given class extends "entity" and adds any attributes or methods relevant to it; for example, the demolitionist class adds its grenade and cover destruction abilities, along with the damage and cooldown of each where relevant, and an addition to the refresh function which handles these new statistics. A class inheritance diagram for this implementation is included below. This allows for greater coding flexibility and extensibility, as any unit can be treated as a generic entity in cases which do not concern its class-specific abilities and can easily be identified as and cast to its specific class in cases which require this. A further modular addition could connect classes to an interface which handles this conversion, removing the need to cast to and mention specific classes in the game logic class.

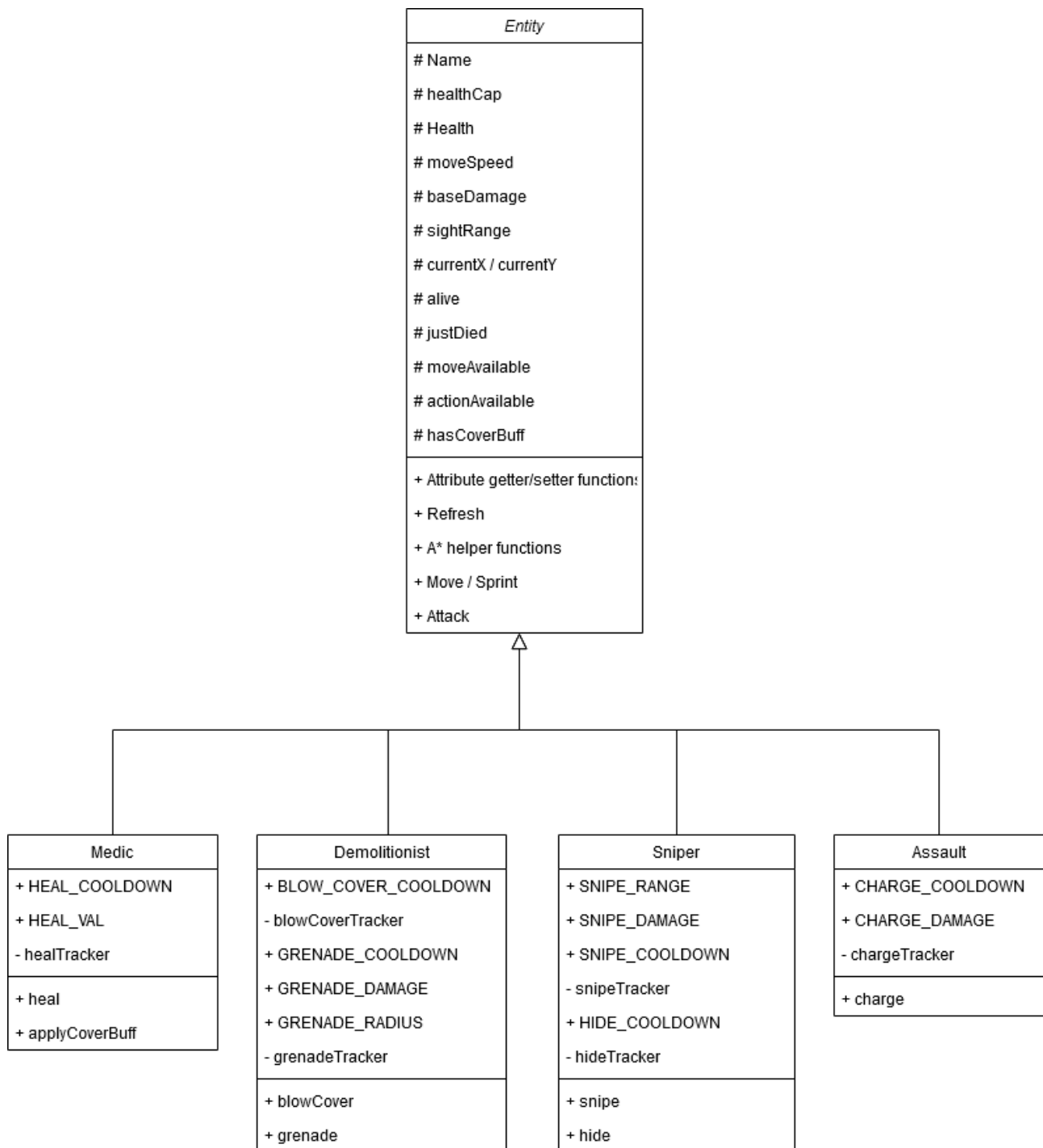


Figure 2: Class inheritance diagram for unit types. Note that, although not explicitly shown, each subclass also overrides the *refresh* function to both invoke the parent class version and update its own ability trackers as well.

## Teamwork model & argumentation

Initial plans for the agent model focused on creating a SOAR cognitive model using the JSOAR library which could understand the game state and actions available to an agent and undertake a comprehensive planning and evaluation step, based on formal logic checks and productions not dissimilar to those in a context-free grammar. These productions would add new pieces of derived information, or new actions to be considered, to the model's memory based on what information is already present in said memory. This approach would have eliminated the need to write my own planning algorithm; I would merely have needed to ensure the model has proper input and output capabilities connected to the game, and that it contains all the logical rules needed to identify and

compare possible moves. Unfortunately, due to a lack of comprehensive documentation or tutorial material, the JSOAR library proved to be too difficult to implement, and as such I opted to implement my own agent system instead.

My model relies primarily on pre-written numerical heuristics in order to assign all of its available moves with a perceived value. Each type of move is assigned a static base value which describes the rough priority a given move has compared to others in a vacuum. Then, the agent class invokes a set of simple logical decisions relevant to a given move which, if met, increment or decrement this base value based on whether they make that move more or less preferable. For example, an attack is already a relatively high priority move, which increases in value if the agent detects that the attack has a high chance of hitting (in two levels, one activating in the range of a 50-75% hit chance and another, higher value being applied if the hit chance is >75%), and increases significantly in value if the opposing unit would be killed by that attack.

This is modified by a simple personality system wherein agents are categorised as reckless, cautious or neutral, and likewise, certain heuristic value adjustments are categorised as aggressive or defensive. In cases where these traits match, for example a more reckless agent considering an aggressive value adjustment, the added heuristic value is multiplied, and likewise in a case where the traits are mismatched, the added value is reduced. This effects a system in which reckless agents more strongly and frequently consider aggressive courses of action and vice versa, introducing more variety into the team's decision making and discussion processes. For example, when evaluating the move command, agents are programmed to increase the value of "safe" positions with lots of cover, so that there is a reduced chance for enemy agents to successfully hit them. However, they are also programmed to significantly increase the value of moves that bring them closer to enemy units when they are not currently close enough to interact with (i.e. attack) them, and this value increase is categorised as aggressive. This in turn should cause more cautious agents to approach enemy units slowly and put more emphasis on keeping themselves safe, while reckless agents will attempt to approach rapidly with less regard for their own safety.

Once an agent has identified and evaluated all moves currently available to it, it selects the one with the highest heuristic value and reports it back to the main game logic process. (Individual agents carry out this process in separate threads to speed up the planning phase; the agent class itself is a runnable, with a helper function to separately update the game state, which is stored in the class using local variables.) Once each agent has done this, the discussion phase occurs, in which each agent presents their highest value move (and articulates their reasons for selecting it to the user as part of the NLG system). Once this is done, each *other* agent on the team applies their own heuristic evaluation to this presented move as if they were the ones intending to make it, adding or subtracting the heuristic value they perceive the move to have.

Once this process has been carried out for each proposed move, the list of moves is sorted based on the newly updated heuristic value of each move and returned to the main function, which then carries out the highest value (legal) move. This allows for a very simple dialogue between agents; a

move is presented along with the reasons why an agent believes that move should be selected, and each other agent presents, in a programmatic sense at least, their own reasons why they believe that move should or should not be selected, and the most broadly supported and valuable move is the one agreed upon by the team. The communication protocol for this dialogue is basic, as proposing agents have no way to dispute or even consider the advice of their peers; rather, the value of their proposed move is automatically adjusted as if they had accepted this input completely. Additionally, only the single top move each agent identifies is up for discussion; no proposing agent will present an alternative move, nor will any evaluating agent suggest a potentially better move, et cetera.

## Natural Language Generation

The heuristic checks mentioned in the previous section also append a flag to the proposed command, which serves to identify that said check has been successfully applied to the move in question. At the discussion stage, wherein agents propose their best actions to the team and expose their reasoning process to the user, these flags are then rendered into natural language. In this case, this is done by directly converting the flag into a pre-written sentence appropriate to that context, which, although rough, *approximates* the behaviour and output of a context sensitive grammar.

Each agent's "personality" also corresponds to a voice code, which causes a different set of sentences which comprise a different character and tone of voice appropriate to this personality. For example, a reckless agent is intended to sound aggressive and excitable, while a cautious agent is intended to be calmer and more reserved. Each voice also has two possible sentences corresponding to a single heuristic flag; together, these features are meant to introduce as much variety into the agents' dialogue as possible, with the intent being to emulate organic natural language dialogue and create the illusion of human characters attached to these units/agents.

For example, say that a demolitionist agent sends a command to the discussion step, intending to throw a grenade that will hit at least two enemy units and kill one. Each command object holds a reference to its parent agent and their personality type, which is extracted by the discussion function and causes it to select from one of three possible switch statements, containing all the relevant sentence options for a given voice code. For each heuristic flag attached to the command, the relevant switch statement is polled and one of the possible options for that flag is retrieved and printed, repeating until every point in favour of that action has been stated. For the grenade example provided, the cautious personality will usually point out that this is an efficient, tactically sound idea, often quoting made-up rules of engagement, while a reckless voice will express their own excitement over the prospect of taking this action.

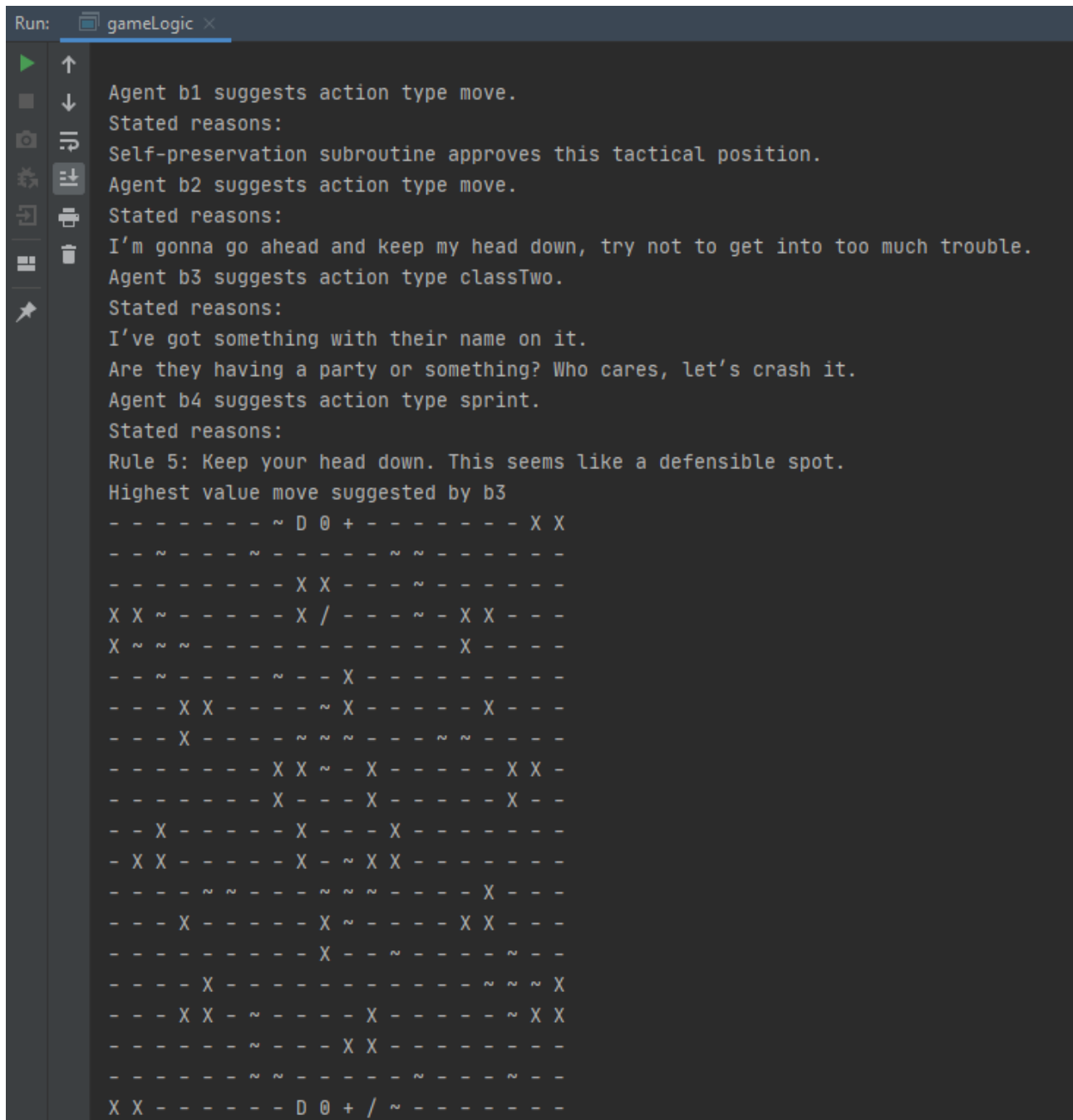
This is, however, a limited implementation of some of the ideas that were planned for the natural language generation component earlier in the project. I had made plans to take advantage of a realization engine or a more advanced CSG system. Instead of relying on pre-written sentences, such an implementation would have a set of grammatical rules and a lexicon of valid words with which to construct new dialogue on the fly. This system could be further customised with a separate set of rules and a separate lexicon for individual agent personalities or voices, all with the intent on introducing more variety into the agents' speech and making their interactions seem more organic.



# Testing and Evaluation

## Personal testing and Example Output

The current implementation of the program runs in two main modes, controlled by an internal boolean variable labelled “demonstration”. With demonstration mode off, the game randomly generates a full 20x20 map and populates both teams with one member of each class. This is intended to be the “full” version of the game and demonstrates the full extent of agent communication and user-facing dialogue, as exemplified in the screenshot below.

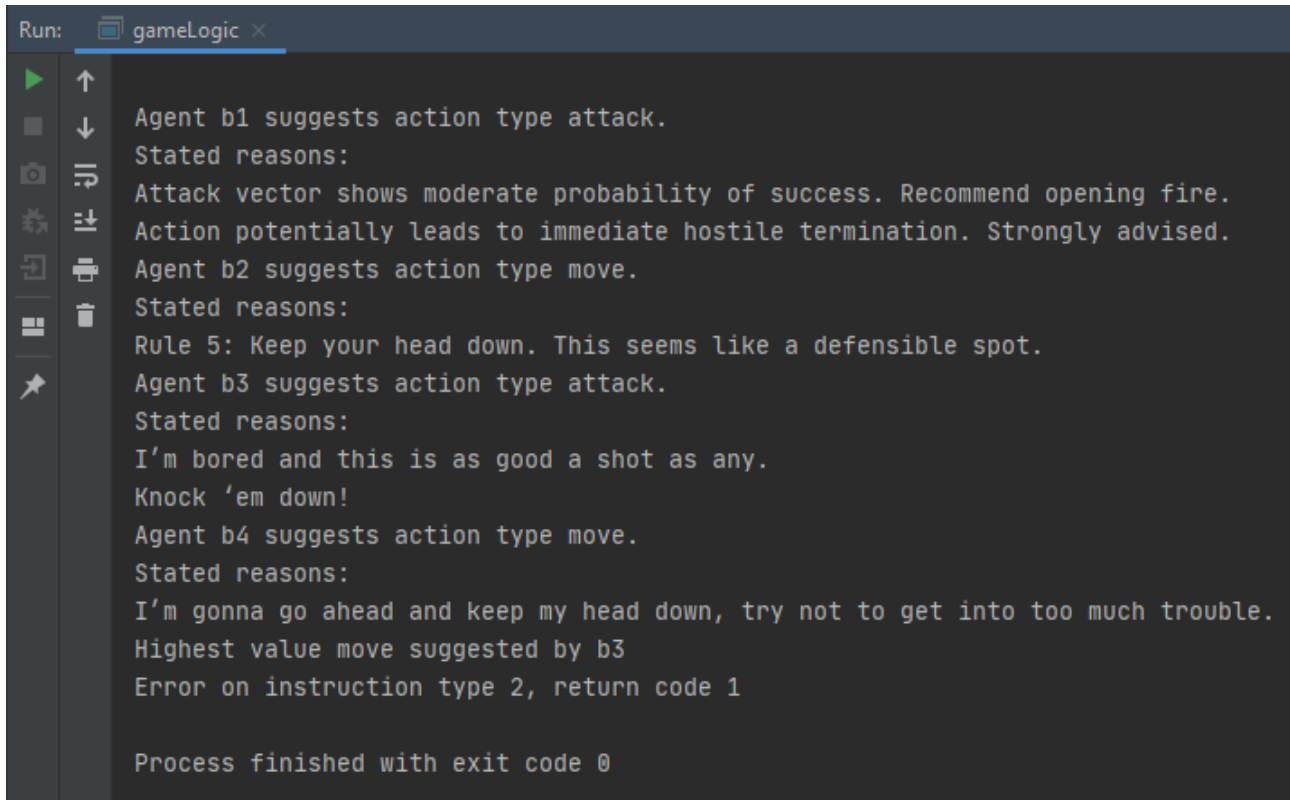


```
Run: gameLogic x
Agent b1 suggests action type move.
Stated reasons:
Self-preservation subroutine approves this tactical position.
Agent b2 suggests action type move.
Stated reasons:
I'm gonna go ahead and keep my head down, try not to get into too much trouble.
Agent b3 suggests action type classTwo.
Stated reasons:
I've got something with their name on it.
Are they having a party or something? Who cares, let's crash it.
Agent b4 suggests action type sprint.
Stated reasons:
Rule 5: Keep your head down. This seems like a defensible spot.
Highest value move suggested by b3
- - - - - ~ D 0 + - - - - - X X
- ~ - - - ~ - - - - ~ ~ - - - -
- - - - - - X X - - - ~ - - - -
X X ~ - - - - X / - - - ~ - X X - -
X ~ ~ ~ - - - - - - - - X - - - -
- ~ - - - - ~ - - X - - - - - -
- - - X X - - - - ~ X - - - - X - -
- - - X - - - - ~ ~ ~ - - - ~ ~ - -
- - - - - - X X ~ - X - - - - X X -
- - - - - - X - - - X - - - - X - -
- - X - - - - X - - - X - - - -
- X X - - - - X - ~ X X - - - -
- - - ~ ~ - - - ~ ~ ~ - - - X - -
- - - X - - - - X ~ - - - X X - -
- - - - - - - X - - ~ - - - ~ - -
- - - - X - - - - - - - - ~ ~ ~ X
- - - X X - ~ - - - - X - - - - ~ X X
- - - - - ~ - - - X X - - - - -
- - - - - ~ ~ - - - - ~ - - - ~ -
X X - - - - - D 0 + / ~ - - - - -
```

Figure 3: Sample output from the demonstration = false version of the game.

This version, however, encounters an unexpected error which appears to be related to the `getLiveEntities` function in the `gameLogic` class. Despite this function being intended to filter out dead units such that they are not considered valid targets when agents attempt to identify legal

moves, some dead units are still being inserted into the game state variables used as part of this process. As a result, this is causing agents to attempt to interact with dead units, causing the sanity checks present in the entity class' functions to fail, which in turn causes the program to exit without reaching the game's actual end state as illustrated below. While this error is potentially easily addressed, either by identifying the issue with `getLiveEntities` or by considering alternative or backup commands if the preferred command is invalid, I was unable to implement these fixes prior to submission.



```
Run: gameLogic x
Agent b1 suggests action type attack.
Stated reasons:
Attack vector shows moderate probability of success. Recommend opening fire.
Action potentially leads to immediate hostile termination. Strongly advised.
Agent b2 suggests action type move.
Stated reasons:
Rule 5: Keep your head down. This seems like a defensible spot.
Agent b3 suggests action type attack.
Stated reasons:
I'm bored and this is as good a shot as any.
Knock 'em down!
Agent b4 suggests action type move.
Stated reasons:
I'm gonna go ahead and keep my head down, try not to get into too much trouble.
Highest value move suggested by b3
Error on instruction type 2, return code 1

Process finished with exit code 0
```

Figure 4: Further sample output from the demonstration = false version, showing the early exit induced by an invalid command.

To alleviate this and demonstrate the intended behaviour of the game, I have included the demonstration mode I used for testing and demonstrating some new features to my supervisors. This version of the game uses a smaller, pre-made map and only one agent on either team, one of which is programmed to start the game with reduced health as a form of illustrating a part of the game as it would take place *in medias res*. This version of the game illustrates the intended end state of the game that takes place when one team has been completely eliminated, i.e. it has no active units left.

```
Run: gameLogic x
-- -- -- / -- --
-- -- -- -- --
Agent b1 suggests action type classOne.
Stated reasons:
High priority tactical decision identified. Context: elimination of enemy unit.
Highest value move suggested by b1
-- -- -- -- --
-- -- -- -- --
-- -- -- -- --
-- -- -- -- --
-- -- -- 0 -- --
-- - X X / X X - -
-- -- -- -- --
-- -- -- -- --

Game over
Team one wins.

Process finished with exit code 0
```

Figure 5: Sample output from the demonstration = true version of the game.

This version of the game always results in the same general outcome, i.e. agent b1 eliminates agent r1 on the opposite team and wins the game. However, it is not entirely deterministic, as I have observed agents b1 and r1 occupying different positions in different executions of this version of the program. This variation, as best I can tell, results from permutations in the agent move evaluation step and in the A\* algorithm used to handle agent b1’s charge attack (as agent b1 is an assault unit). For example, agent r1 may identify two equally preferable locations to move through and move to one or the other in different instances of the program based on which one was created first.

Besides this output analysis and the ongoing testing which occurred as part of the Agile development process, I was unfortunately unable to conduct further testing and output analysis on the program, due to both the short span of time I had between finishing an acceptable artefact for submission and the submission deadline, and the fact that the aforementioned bug with the non-demonstration version was only identified around this time. Besides analysing the behaviour of the agents for determinism or lack thereof based on fixed game start states, I had also made some plans to analyse the efficiency of agents’ behaviour by way of charting the number of turns required to finish a game, based on the size and complexity of the map and number of units involved.

## Conclusion

As mentioned in the introduction to this report, I believe this project successfully implements all of the stated primary objectives I established at the beginning of the academic year, and at least some of the secondary objectives. I must stress, however, that it does so in a basic fashion; the provided artefact is perfectly suitable as a proof of concept for the ideas discussed as part of my introduction, but it does so in a somewhat limited fashion. For the most part, I am aware of these limitations, as

they were either consciously included in the higher level design of the project, or because I identified them as development continued but did not have the time to address them. In both cases, such limitations predominantly arose due to time constraints or imposed by working on the project alone and being subject to a submission deadline, or due to my planning around said time constraints. While I am overall satisfied with the implementation, as it serves its required purpose of demonstrating the core ideas of the project, I will briefly outline some of the main limitations I have identified and discuss potential extensions to the project that would address them.

As discussed earlier, the agents included in this implementation have a limited communication protocol which only allows for the value of already proposed commands to be altered. While this is sufficient to achieve the stated objectives for agent communication, a more refined model might allow for counter-proposals; such a system would allow one agent to present an action it intends to take, and in response, another agent would state an alternative action it believes that the proposing agent should take instead, which could likewise be subject to evaluation by the proposing agent and the rest of the team. Such a model would allow for more extended and comprehensive communication between agents, and hence richer dialogue for the user.

Similarly, while their current behaviour is acceptable, agents would behave with a greater degree of intelligence (as far as a potential user would see) if they could plan sequences or combinations of actions. To give one example, it would be ideal if two agents could explicitly co-ordinate and agree to have one demolitionist destroy a piece of cover that is blocking another allied unit's line of sight to an enemy, thus allowing them to make a potentially effective attack. While it is possible for such a sequence of actions to arise in the current model, such an occurrence is mostly coincidental; it would be preferable if the agents specifically and "consciously" intended for such an outcome. And while I have made some attempts to add more value to moves which would bring an agent closer to being able to interact with enemy units, there is no proper system for evaluating the potential long term value of a move; i.e. the agent does not take into account the value of the potential successive moves or game state resulting from the current moves available to it, making it potentially vulnerable to local maxima. For example, an agent may have two move actions available to it wherein one has a higher perceived value on its own, but after taking that action, the agent only has very low value moves available to it, while the other move may have a lower initial value but enable an action with a much higher value than any one possible in the other move tree. This could potentially be resolved by constructing an explicit decision graph for the agent's actions and applying an iterative deepening depth-first search algorithm, though this is a potentially computationally expensive solution and not the only possible one.

Likewise, though I have worked to ensure a decent level of variety for the agents' natural language output, the approach of having manually pre-written sentences or grammars limits the amount of available dialogue options based on how many such sentences I, or any person in general working on a similar feature, am willing or able to add on my own. One could potentially create a much more varied dialogue bank by taking advantage of AI language models such as GPT-3; such an approach would allow an extremely large amount of varied dialogue to be produced for minimal effort beyond finding suitable input material and parameters for the model, the output of which could then be curated and inserted into the game's code. Such curation would likely be preferable to directly piping the model's output to the end user, as it would allow for some developer oversight which could prevent unexpected or inappropriate text from being presented alongside more suitable

dialogue options. Models on the scale of GPT-3 are typically resource intensive to the point of requiring research-level computers to run, but it may be possible to access them via other means, such as by working with the AI Dungeon project provided by the Latitude team.

## Appendix

### A: Bibliography

1. Hill, Randall W., Johnny Chen, Jonathan Gratch, Paul Rosenbloom, and Milind Tambe. "Intelligent agents for the synthetic battlefield: A company of rotary wing aircraft." In *AAAI/IAAI*, pp. 1006-1012. 1997.
2. Kitano, Hiroaki, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru Asada. "The RoboCup synthetic agent challenge 97." In *Robot Soccer World Cup*, pp. 62-73. Springer, Berlin, Heidelberg, 1997.
3. Anderson, John R. "Cognitive Architectures Including ACT-R." *Cognitive Studies: Bulletin of the Japanese Cognitive Science Society* 26, no. 3 (2019): 295-296.
4. Çelikok, Mustafa Mert, Tomi Peltola, Pedram Daei, and Samuel Kaski. "Interactive AI with a Theory of Mind." *arXiv preprint arXiv:1912.05284* (2019).
5. Egges, Arjan, Sumedha Kshirsagar, and Nadia Magnenat-Thalmann. "A model for personality and emotion simulation." In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 453-461. Springer, Berlin, Heidelberg, 2003.
6. Black, Elizabeth, and Katie Atkinson. "Dialogues that account for different perspectives in collaborative argumentation." In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 867-874. 2009.
7. Tambe, Milind, and Weixiong Zhang. "Towards flexible teamwork in persistent teams." *Autonomous Agents and Multi-Agent Systems* 3, no. 2 (2000): 159-183.
8. Becker, Raphen. *Exploiting structure in decentralized markov decision processes*. University of Massachusetts Amherst, 2006.
9. Schurr, Nathan, Steven Okamoto, Rajiv T. Maheswaran, Paul Scerri, and Milind Tambe. "Evolution of a teamwork model." (2005).
10. Shmoys, David B., and Éva Tardos. "An approximation algorithm for the generalized assignment problem." *Mathematical programming* 62, no. 1-3 (1993): 461-474.
11. Brill, Zachary M., and Christopher L. Dancy. "Simulating Human-AI Collaboration with ACT-R and Project Malmo." (2018).
12. Bansal, Gagan, Besmira Nushi, Ece Kamar, Eric Horvitz, and Daniel S. Weld. "Optimizing AI for Teamwork." *arXiv preprint arXiv:2004.13102* (2020).
13. Madden, Neil. "Collaborative narrative generation in persistent virtual environments." PhD diss., University of Nottingham, 2009.

14. Gatt, Albert, and Emiel Krahmer. "Survey of the state of the art in natural language generation: Core tasks, applications and evaluation." *Journal of Artificial Intelligence Research* 61 (2018): 65-170.
15. Lemon, Oliver. "Adaptive natural language generation in dialogue using reinforcement learning." *Proc. SEM-dial* (2008): 141-148.
16. Gatt, Albert, and Ehud Reiter. "SimpleNLG: A realisation engine for practical applications." In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pp. 90-93. 2009.
17. Reiter, Ehud, Somayajulu Sripada, Jim Hunter, Jin Yu, and Ian Davy. "Choosing words in computer-generated weather forecasts." *Artificial Intelligence* 167, no. 1-2 (2005): 137-169.
18. Zhong, Qiaoting, Xiuyi Fan, Francesca Toni, and Xudong Luo. "Explaining Best Decisions via Argumentation." In *ECSI*, pp. 224-237. 2014.
19. Caminada, Martin W., Roman Kutlak, Nir Oren, and Wamberto Weber Vasconcelos. "Scrutable plan enactment via argumentation and natural language generation." In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pp. 1625-1626. 2014.

## B: Initial Time Plan

- Weeks 1-2: Initial planning (formulating objectives, scheduling meetings, writing and submitting DOER)
  - Weeks 3-4: Context survey work
  - Weeks 5-6: Game prototyping (Majority of game implementation, including level generation, base game mechanics/logic, unit classes and actions should all be added by the end of this stage)
- Weeks 7-8: Base agent model implementation (To include reasonable understanding of game mechanics and strategy, planning and communication/argumentation layer. Personality traits are desirable at this stage but not required beyond what is required to allow for meaningful argumentation.)
- Weeks 9-10: Work on personality system and reinforce any lacking areas in planning and game understanding.
- Weeks 11-12: NLG translation system. (To be able to produce readable output, regardless of repetitiveness, by prototype submission date. Also aim to conduct or complete artefact evaluation by this time, if evaluation with volunteers is being conducted.
- Weeks 13-14: Refine NLG system for dialogue variety and personality variance. Bug testing can be done alongside features as they are implemented but this may be an ideal spot for a dedicated bug testing/fixing period.
- End of semester to submission: Use remaining time to refine or implement any remaining features, depending on Christmas holiday plans.
  - 1-2 weeks prior to final submission: Complete full report.

## C: Program setup

I used IntelliJ Idea with a Java version 15 SDK to write and run the program. A copy of the project folder has been included in the final submission, and my personal recommendation is to import it into IntelliJ and use that to run it as well. This folder is mislabelled “CS5099Game” but is original to and fully intended for this project. I used Maven to import several external libraries which are listed here in case they are not properly imported alongside the project code.

