HÖGSKOLAN SKÖVDE

Institutionen för kommunikation och information
Examensarbete i datalogi 30hp
C-nivå
Vårterminen 2008

Threat Analysis Using Goal-Oriented Action Planning

Planning in the Light of Information Fusion

Philip Bjarnolf

Threat Analysis Using Goal-Oriented Action Planning

Submitted by Philip Bjarnolf to the University of Skövde as a dissertation towards the degree of B.Sc. by examination and dissertation in the School of Humanities and Informatics.

May 27, 2008

I hereby certify that all material in this dissertation which is not my own work has been identified and that no work is included for which a degree has already been conferred on me.

Signature: ______________________________

Threat Analysis Using Goal-Oriented Action Planning

Philip Bjarnolf

Abstract

An entity capable of assessing its and others action capabilities possess the power to predict how the involved entities may change their world. Through this knowledge and higher level of situation awareness, the assessing entity may choose the actions that have the most suitable effect, resulting in that entity's desired world state.

This thesis covers aspects and concepts of an arbitrary planning system and presents a threat analyzer architecture built on the novel planning system Goal-Oriented Action Planning (GOAP). This planning system has been suggested for an application for improved missile route planning and targeting, as well as being applied in contemporary computer games such as *F.E.A.R. – First Encounter Assault Recon* and *S.T.A.L.K.E.R.: Shadow of Chernobyl*. The GOAP architecture realized in this project is utilized by two agents that perform action planning to reach their desired world states. One of the agents employs a modified GOAP planner used as a threat analyzer in order to determine what threat level the adversary agent constitutes. This project does also introduce a conceptual schema of a general planning system that considers orders, doctrine and style; as well as a schema depicting an agent system using a blackboard in conjunction with the OODA-loop.

Keywords: Threat Analysis, Goal-Oriented Action Planning, STRIPS, Planning Systems, Decision Support, AI, OODA-loop, Coalition Battle Management Language, Commander's Intent, Effects Based Planning, Information Fusion

Table of Contents

1 Introduction 1

1.1 Thesis outline.. 2

2 Background 3

2.1 Situation awareness.. 3

2.2 Components and aspects of a planning system 3

2.2.1 Finite state machine.. 5

2.2.2 Blackboard.. 6

2.2.3 A blackboard in conjunction with the OODA-loop 8

2.2.4 Message passing.. 8

2.2.5 Intent and the advent of C-BML.. 9

2.2.6 Procedural tactics planning in a dynamic environment 11

2.2.7 Goal-Oriented Action Planning 11

2.2.8 Regressive searches.. 17

2.2.9 Effect based planning.. 18

2.2.10 Hierarchical Task Network.. 19

2.3 Multi-agent and squad behavior 20

2.3.1 Centralized vs. decentralized approach 21

3 Problem 23

4 Methods 25

4.1 Method for objective 1: Development of the necessary components............... 25

4.2 Method for objective 2: Evaluation of gathered data and observed behavior... 25

5 Realization.. 26

5.1 Overview 26

5.2 Identified actions and goals.. 29

5.3 Identified world state keys 30

5.4 Further details of the architecture.. 31

5.5 Scenario rules and conditions.. 34

5.6 Class diagram 35

6 Results 37

6.1 Results from objective 1: Development of the necessary components............ 37

6.2 Results from objective 2: Evaluation of gathered data and observed behavior 39

6.2.1 Analysis of data log excerpts.. 39

6.2.2 A walk-through of an arbitrary round 40

7 Conclusion 43

7.1 Research discussion 43

7.2 Future work 44

Acknowledgements 45

References 46

Appendix

1 Introduction

Threat assessment is a vital part in everyday life as it enables us to prevent or evade dangerous or otherwise unwanted events. Through threat assessment we evaluate the elements of our world as we perceive and conceive it. These elements can be seen as key building blocks of the world, and depending on the situation, more or less of these keys have a direct impact on the outcome of a certain scenario taking place. In the eyes of the beholder, or even better in the mind of the interpreter, the relevant key parts of the world are registered and in one way or the other given a certain value or reference.

As our world, depending on the level of detail we wish to apply, has what seems to be an almost endless amount of world states, it would be impractical, unnecessary or even impossible to keep track of and model each and every world state. It therefore seems intuitive and wise to only model those world states deemed necessary for the scenario or context at hand.

Once the relevant world state keys have been identified their paired value may be changed by an arbitrary *action* that causes a world state transformation to occur, that is to say that the action has an *effect*. An entity capable of an arbitrary set of actions is by default only capable of changing the world states according to the effects of those available actions. However, this is not to say that the entity is unable to realize world states it has not planned for. Since it is perfectly possible to perform an action without knowing all of its consequences, i.e. effects or world state transformations, an entity may very well experience how things unfold in unseen and sometimes unwanted ways. This also means that in order for a threat assessing entity to comprehend a threat, the entity must be able to picture how a series of actions could realize an unwanted world state. An entity's insight of its and others action capabilities is thus tightly coupled with the entity's threat analysis capabilities.

This concept is heavily used in this thesis which aims to analyze and develop a threat analysis planning architecture built on Goal-Oriented Action Planning (GOAP). By slightly modifying and thereby extending the GOAP architecture, a higher level of situation awareness may be obtained; enabling entities using a threat analyzer to employ better strategies and decisions. In this project, two agents utilize GOAP technology for their action planning, where one agent also uses a modified planner acting as a threat analyzer, capable of assessing its adversary's threat level in the simulated scenario. The intended audience of this work is people having interest in AI technologies in the field of planning systems and computer generated forces.

GOAP is an AI technology well suited to dynamic environments such as military operations (Doris & Silvia, 2007), and is used in the area of planning to among others extend the simple use of finite state machines (FSMs). GOAP has successfully been employed in contemporary games like *F.E.A.R. – First Encounter Assault Recon* and *S.T.A.L.K.E.R.: Shadow of Chernobyl*, and a recent research experiment has found that GOAP for many reasons is a superior AI system, compared with just a regular FSM architecture (Long, 2007). The use of real-time action planning improved the process of developing character behaviors in F.E.A.R. and was an attempt to solve the complexity caused by the combination and interaction of all occurring behaviors (Orkin 2006). Furthermore, the Naval Postgraduate School in Monterrey, USA, suggested GOAP for an application for improved missile route planning and targeting (Doris & Silvia, 2007).

1.1 Thesis outline

This report is structured as follows:

Chapter 2 describes relevant background of the project's problem domain where components such as situation awareness and blackboards may play a vital part, as well as multi-agent behavior and aspects of message passing.

Chapter 3 contains the problem description and statement where this project's aims and objectives are described.

Chapter 4 discusses the methods at hand and why a certain method was chosen, and how it is supposed to be carried out.

Chapter 5 describes the realization of the project's implementations, from overview to detailed design decisions and class diagrams.

Chapter 6 presents the results and an analysis of what has been observed during the implementation and execution of the architecture, according to the objectives in chapter 3.

Chapter 7 holds a conclusion and research discussion of the project, as well as suggestions for future work.

2 Background

In this section relevant areas of the project's problem domain are described. Section 2.1 deals with the basic concept of situation awareness while section 2.2 and its subparts discuss the components and aspects of a planning system, such as finite state machines, blackboards, the OODA-loop, procedural tactics, message passing techniques and Goal-Oriented Action Planning (GOAP). Section 2.3 shortly deals with multi-agent and squad behavior in the light of a centralized and decentralized planning approach.

This report shares the definition of an *agent* and *emergence* with that of Holland (2008). This outlook sees (software) agents as "…programming constructs that maintain their own rule-base and instantiate themselves interactively within the software environment." (Holland, 2008, pp. 1).

2.1 Situation awareness

How decision making agents perceive and conceive their environment and the "knowing of what is going on" (Endsley, 1995, pp. 37) can be referred to as *situation awareness* (SA), which is highly involved with how well an agent is able to cope with its environmental challenges. For example, a blind and deaf agent that participates in a game of hide and seek, has far lower SA than an unimpaired counterpart; who on the other hand, has lower SA than an agent that can communicate with an airborne teammate who shares its bird eye view of the situation. This plain example shows how an agent's SA can be increased by sharing the information of the situation at hand to other agents. The definition of SA and its build-up can be theoretically explained or thought of in numerous ways, as done by Lagervik & Gustavsson (2006), Endsley (1995), Ackoff (1999), and Bedney & Meister (1999). However, the main parts involved are without doubt the agent's sensory system, its memory, and its ability to interpret, understand, and mediate to others its subjective impressions and gathered knowledge.

By centralizing this SA or knowledge through a *blackboard*, as suggested by Orkin (2006), all participating agents become an entity with the same basic world comprehension, even though individual *styles* may interpret the information in different ways.

2.2 Components and aspects of a planning system

A planning system may contain an arbitrary amount of components and this chapter will only consider and discuss some fundamental of them. Figure 1 introduces a conceptual schema of an arbitrary planning system based on all references regarded in this thesis. The schema shows how an agent's style influences the inputs from the world, broken up into sensed orders and world states. As the sensed data is passing through the personality layer, it gets styled according to the agent's personality, after which the brain, i.e. reasoning and planning system, is able to draw a conclusion and formulate the most promising plan; which in turn changes the world into or towards a desired state.

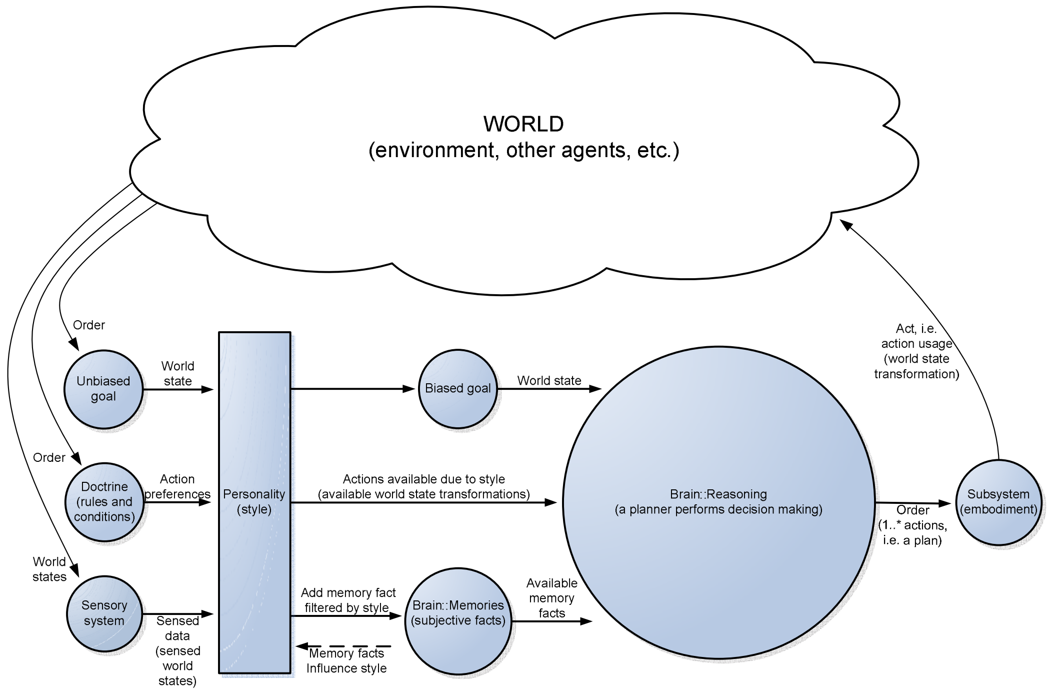

WORLD
(environment, other agents, etc.)
Order
Unbiased goal
World state
Order
Doctrine (rules and conditions)
Action preferences
World states
Sensory system
Sensed data (sensed world states)
Personality (style)
Biased goal
World state
Actions available due to style (available world state transformations)
Add memory fact filtered by style
Memory facts Influence style
Brain::Memories (subjective facts)
Available memory facts
Brain::Reasoning (a planner performs decision making)
Order (1..* actions, i.e. a plan)
Subsystem (embodiment)
Act, i.e. action usage (world state transformation)

Figure 1. A conceptual schema overview of an arbitrary planning system.

All actions undertaken by agents are consequences of a desire to achieve certain goals. As these goals or tasks might have been proposed by another commanding agent, the medium or technique mediating the task has a great responsibility to ensure that it is not altered or perceived in an inaccurate way by the agent supposed to execute the task (i.e. the taskee). By having the commander only telling *what* he wants to be done instead of *how*, the taskee is given the authority to execute (and formulate) a plan of its own (Kleiner, Carey & Beach, 1998). The concept of a correct picture of the task or desired end-state, as first formulated by the commander, is referred to as Commander's Intent (CI), which is described in more detail in section 2.2.5.

In order to achieve a goal, a *plan* consisting of actions has to be formulated and executed by the agent. The plan is thus nothing more than a set of actions that has to be executed in a correct order with certain timing, consequently causing one or more *effects* which results in an alteration of a world state. In Figure 1 the plan formulating system, i.e. the reasoning brain of an agent, receives styled data input from the world. Depending on the current goal, actions at hand, and past experiences, the plan formulating system outputs an action or a series of actions (i.e. a plan) that are realized by a subsystem of the agent; consequently changing a world state towards or into the desired end-state.

The plan formulating system is called a *planner*, and is in other words allowing agents to *formulate their own action sequence* in order to accomplish a certain goal. Orkin (2004c, pp. 222) states that "in order to formulate a plan, the planner must be able to represent the state of the world in a compact and concise form", which incorporates identifying the most important variables of the world where the current scenario is taking place. This means that even though the actual world state consists of a myriad of variables, only the most significant variables should be used in order to keep the planning as simple and fast as possible. The International Game Developers Association (IGDA, 2005) describes a plan as:

> (…) a valid sequence of actions that, when executed in the current state of the world, satisfies some goal or multiple goals. A sequence of actions is valid if each action's preconditions are met at the time of execution. The planner attempts to find an optimal plan, according to some cost metric per action. The planning process cannot manipulate the actual state of the world. Instead the planner operates on a copy of the world state representation that can be modified as the planner evaluates the validity of all possible sequences of actions.

2.2.1 Finite state machine

A finite state machine (FSM) is probably the most basic technique for agent behavior and is used extensively in today's applications where a certain behavior is to be modeled. The actual FSM design and its complexity may vary depending on what is being implemented, but each state is overall more or less connected to other states in a directed graph, and the occurring transitions are predetermined and essentially consist of three parts (Buckland, 2005):

- **Entry**. The `Entry` function is run once each time the state is loaded into the state machine. This is where necessary initialization occurs, e.g. initialization of the state's variables or variables belonging to the owner of the state machine.
- **Execute**. Once the state has been initialized by the `Entry` function, the `Execute` function takes over and is run until the state is to be replaced by a new state.
- **Exit**. When the currently loaded state is to be replaced by a new state, the currently loaded state's `Exit` function is run once. This enables a clean transition from the currently loaded state to its successor, meaning that the state is given an opportunity to tidy up and reset necessary variables.

At any time only one state is loaded into a state machine which executes the state according to the parts above. The states are usually implemented using the Singleton design pattern which results in a lower memory usage than if each unique state were to be instantiated several times for each state machine using it. The possible drawback of this implementation is that no owner[1] specific data can be held in the concerned state; the owner must implement the necessary variables which then can be accessed and used by the current state. Depending on the environment and problem at hand, multiple state machines may be used and even run in parallel or in a hierarchical fashion (HFSM), enabling multiple states to execute at the same time. For example may a soldier agent have its physical and mental conditions simulated in two different state machines. The use of multiple state machines running in parallel may permit more complex behavior, but it may also include the complexity adding up to illegal combinations or unwanted behavior. State machines can be used in many different areas and applications, e.g. the states may be realized as the possible choices of a menu system or the logic of getting dressed and putting the cloth on in a correct order; where it is probably preferable to put on the socks before the shoes etcetera. In the planning domain, a state machine could use a plan by subsequently loading, executing and unloading the states contained within the plan.

[1] The *owner* is the entity or instance that implements the state machine into which states are loaded and later on executed and unloaded.

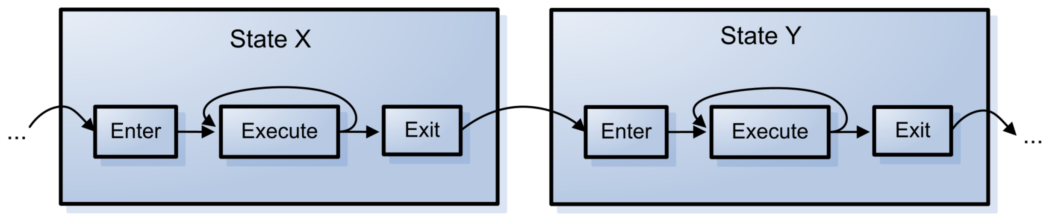


Figure 2. Each state's internal structure is run as state transitions occur.

2.2.2 Blackboard

Through posting observations and intentions, entities[2] utilizing a blackboard may endue an increase of situation awareness, possibly leading to the entity gaining the upper hand of the situation and in the end being on the side that prevails. The entities can be said to act as knowledge sources responsible for handling both the condition-part and action-part, described by Buschmann (1996) as *evaluation* of the current state to determine if a contribution can be made, and *result production* that may cause a change to the blackboard's contents. He also states that "in blackboard several specialized subsystems assemble their knowledge to build a possible partial or approximate solution.". This implicitly means that a blackboard requires several entities accessing and using it in order to useful; a blackboard used by only one entity or subsystem is unproductive and somewhat useless. Furthermore, the accuracy of an agent's notion of a world state thus stems from the squad's gathered and evaluated data, which ought to be the very foundation of optimal decision making. Buschmann (1996, pp. 74) depict a blackboard architecture as a:

> (…) collection of independent programs that work cooperatively on a common data structure. Each program is specialized for solving a particular part of the overall task, and all programs work together on the solution. They do not call each other, nor is there a predetermined sequence for their activation. Instead, the direction taken by the system is mainly determined by the current state of progress. A central control component evaluates the current state of processing and coordinates the specialized programs. This data-directed control regime is referred to as *opportunistic problem solving*.

Orkin (2004b) identifies several advantages of using a centralized blackboard, among others the increase of maintainability as the code ages and scales, and the decrease of overhead as the blackboard acts as an interface for accessing the shared data between multiple agents. This enables the subsystems to be decoupled as the blackboard acts as a central resource, through which various subsystems may handle their communication or direct data handling. Orkin (2004b, pp. 199) speaks of the following experience regarding blackboards:

> Our tight development schedule did not allow for the implementation of a group behavior layer, but we were able to solve the problems (…) by leveraging existing information in our pathfinding, vision, and sensory systems. The addition of a simple blackboard gave us a means to share this information among multiple agents.

---

[2] Here an *entity* may be an arbitrary unit contributing through the blackboard. Hence an entity could be a sensor posting information to a memory, or a military agent posting information to its commander, or an organization posting relevant context data to a knowledge base.

According to Reynolds (2002) the use of globally accessible data, like a blackboard, can be used to avoid lengthy messages being sent to multiple agents. He furthermore explains that this is however linked with trouble during the debugging process so the agents should nevertheless report the change in data, as global data should not be seen as a replacement for messages. The blackboard shown by Orkin (2004b) shows an architecture where *records* are posted, removed, queried and counted at the blackboard by the involved agents. The structure of a record consists of a record type, the ID of the poster and target, and some generic four-byte data. For example when an agent goes prone, it posts the current time to the blackboard in a record of type `kBB_ProneTime`. The blackboard is recognized to facilitate coordinate timing, sharing world objects, and varying actions and animations among multiple agents. This type of blackboard architecture was used in the game *No One Lives Forever 2 : A Spy in H.A.R.M.'s Way*.

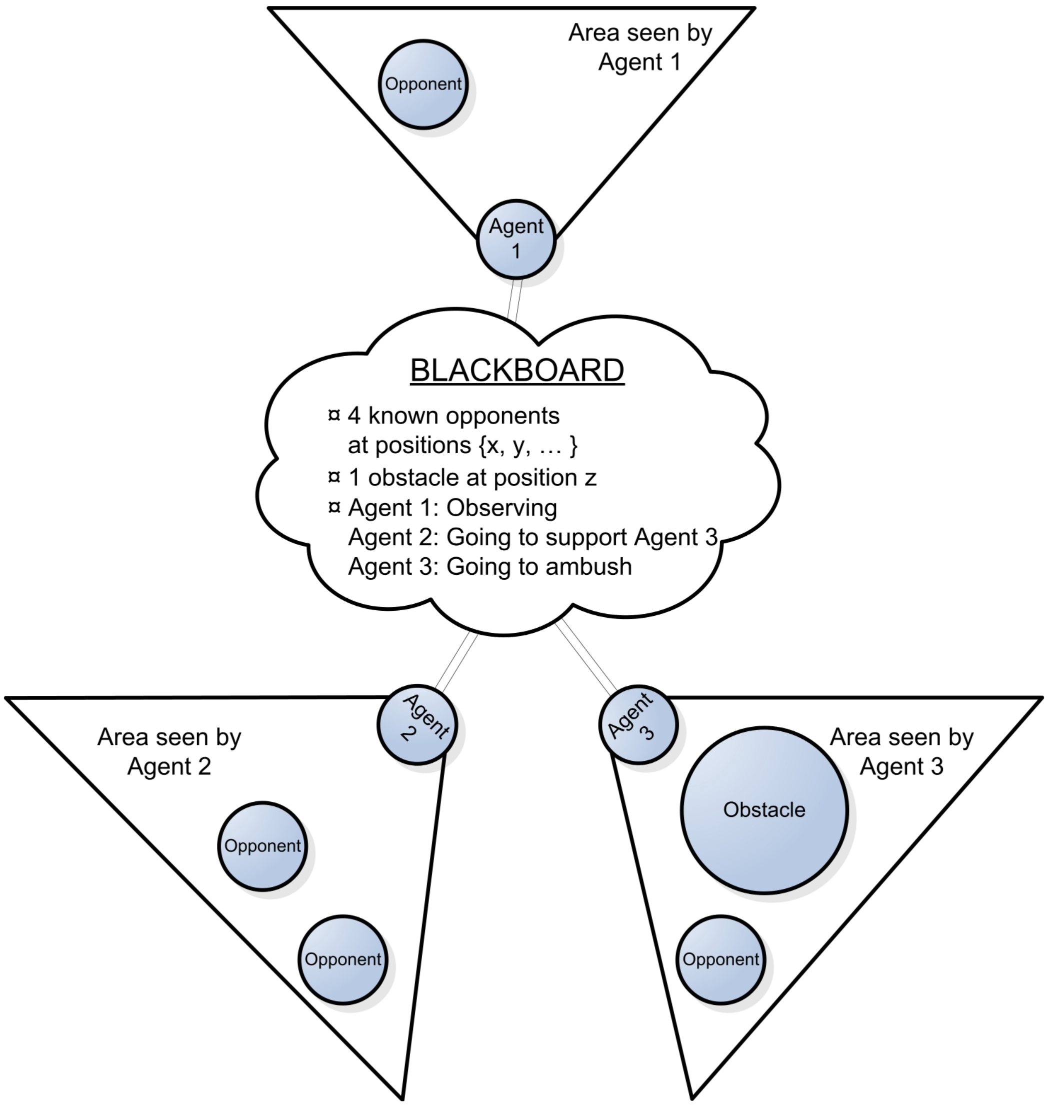


Figure 3. An example of a blackboard acting as a situation awareness repository of gathered knowledge and understandings.

2.2.3 A blackboard in conjunction with the OODA-loop

The U.S. Marines amongst other organizations uses a model called OODA to capture the continuous nature of command and control (Breton & Rousseau, 2005), developed by Col. John Boyd during the Korean War. The model consists of the four phases Observe, Orient, Decide and Act, which all are reiterated in order to produce an optimal action based on the information observed in the environment. By cycling through these steps faster and better than an opponent, the faster entity may be given the upper hand of the situation, thus being the one prevailing. Different variations of the OODA-loop have been presented (Breton & Rousseau, 2005) which may be used to improve the original design. By taking the OODA-loop as a foundation of the decision making and combining it with an agent that uses a blackboard, a picture as depicted by Figure 4 is introduced.

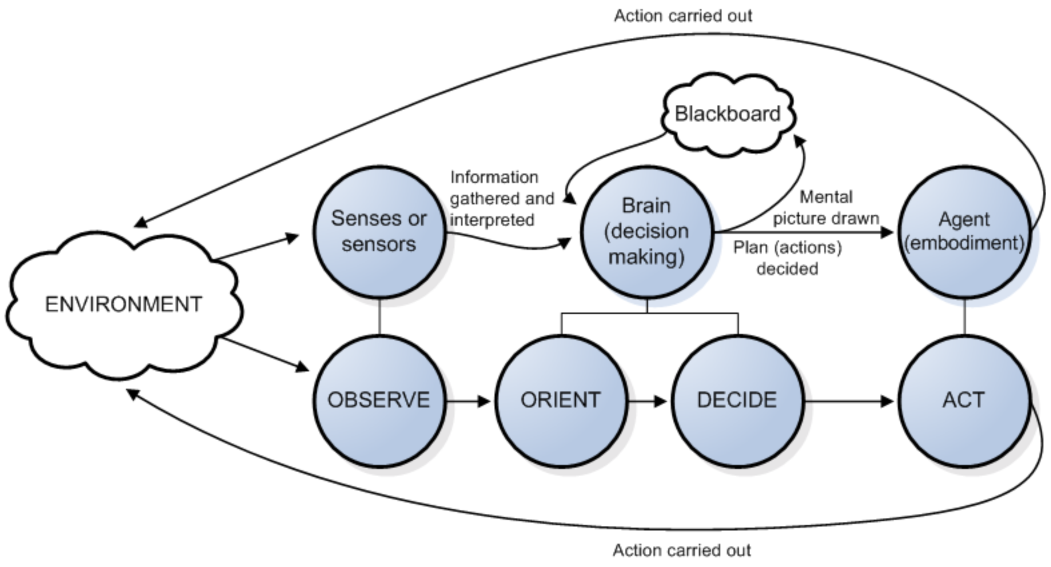


Figure 4. A conceptual schema of an agent system using a blackboard in conjunction with the OODA-loop.

From Figure 4 we see the first phase Observe, in which the environment is comprehended by some sort of senses or sensors. For instance, this could be an agent's eyes seeing an obstacle or the agent's hearing sense sensing a movement at a certain location. The next OODA phases Orient and Decide are handled by the agent's brain, evaluating the newly arrived data while considering what is already known, and by doing this the current plan may be refined and updated. Depending on the input, the brain makes a decision on what the action performer, i.e. the agent's embodiment or subordinates, must do to improve the odds of accomplishing the current goal. Note that an action does not have to be finished before a new OODA iteration can commence; it is fully sufficient that the action is being done under an arbitrary amount of time after which another action might be more favorable. In the hide and seek example this would mean that an agent who is carrying out a simple move action suddenly switches to a hide action in the presence of a threat.

2.2.4 Message passing

Reynolds (2002) states that "The use of message passing is very effective when implementing team-based AI. Using messages to communicate between different layers of the hierarchy is a very natural way of passing orders and information.".

By employing message passing the system benefits from increased information hiding and decoupling of interacting components, since the entity sending the message does not have to know anything about the recipient, other than its ID. By sending messages the interacting entities are also able to share their awareness of the situation, which for instance built the squad's shared tactical picture in the SAMPLE agent architecture (Aykroyd, Harper, Middleton & Hennon, 2002), further described in section 2.3. Van der Sterren (2002a) gives these reasons to why it is more attractive to use messages to pass squad member state information around, than inspecting the AI objects directly:

- You can model communication latency by briefly queuing the messages.
- You can present the message in the game using animations or sound.
- You can filter out messages to the player to prevent overloading him.
- The squad member will send messages to dead squad members it assumes still to be alive, adding to realism (and preventing "perfect knowledge").
- You can use scripted entities to direct one or more squad members by having the scripted entities send messages to these squad members.
- You can accommodate human squad members, whose state information is largely unavailable, but for whom the AI can emulate messages with observations and intentions.

Besides incorporating an agent-to-agent communication system, the message passing system may also serve as a tool for communication between other components of the system. By using messages the system becomes an event-driven architecture, which generally is preferred because of its efficiency and polling avoidance (Buckland, 2005). The event-driven architecture allows an entity to be sent the information of interest and thereafter act upon it as it may seem fit. Yet another benefit is explained by Reynolds (2002, pp. 267):

> Fortunately, this implementation [the use of message passing] offers the programmer the opportunity to log the messages between the levels of command. This can be used to provide quite natural dialogue between levels of command outlining all the information and decisions that have been made. This can be invaluable when tracing where the errant information has come from or which decision was at fault.

2.2.5 Intent and the advent of C-BML

Another aspect of message passing is the use of a consistent language which in the best of worlds would be unambiguous and interpretable by both man and machine. This would enable the message receivers to without subjective influences grasp the sender's ambition of the current subject. The benefits of a common and unambiguous language, fit for human-to-human, human-to-machine, or machine-to-machine interaction, seem almost endless. Alberts (2007) claims that "Changing the language we use will free us in our search for a solution for a better approach. If we develop a suitable language, it will point us in a right direction as well.". Work in this area is currently underway with the evolution of the Coalition Battle Management Language (C-BML) and Tactical Situation Object (TSO), as described by Gustavsson, Nero, Wemmergård, Turnitsa, Korhonen, Evertsson and Garcia (2008). The C-BML standard is based on the 5W, which according to Kleiner et al. (1998, pp. 883) makes it:

> …possible to describe any mission or task given to a subordinate in a standardized manner with a **who** (relates to a task organization database entry), **what** (in doctrinal terms), **when** (specified time, on order, or keyed to a triggered event), **where** (related to a coordinate or graphical control measure), and **why** (in doctrinal terms).

C-BML is supposed to express the knowledge which flows within a certain system like an autonomous agent or automated decision-support system, and unambiguously describe Commander's Intent (CI) without in any way impairing or constraining the CI (Lagervik & Gustavsson, 2006; Blais, Galvin & Hieb, 2005; Gustavsson, Hieb, Eriksson, Moore & Niklasson, 2008). As C-BML stems from BML (Battle Management Language), it shares the same purpose in being able to formulate orders, requests and reports through a standardized language for military communication. In turn, BML is built on the Command and Control Lexical Grammar (C2LG) which was first formalized by Hieb & Schade (2006) and is still under development. Borgers, Spaans, Voogd, Hieb & Bonse (2008) claims that "As long as every agent, operator and C2SS [Command and Control Support System] sends and receives BML messages, a seamless connection between them is possible.". The representation of CI is to contain the mission team's purpose, the anticipated End-State of the mission and desired key tasks (Gustavsson et al., 2008). Furthermore, the intent can be divided into an *explicit* and *implicit* part, where the explicit intent can be seen as the commander's straightforward orders, while the implicit intent is developed over a longer time, prior to the mission, and consist of the *expressives*[3] and the concepts, policies, laws and doctrine agreed to by military, civil, organizations, agencies, nations and coalitions (Gustavsson et al., 2008). Farrell (2004) gives the example that if explicit intent is "*to capture the hill*", then implicit intent might be "*to capture the hill with minimal battle damage*" or "*to capture the hill with Air Force assets only*". Figure 5 depicts the Operations Intent and Effects Model (OIEM) that simulation systems can use for both forward and backtracking planning. The model shows how an initial state is detected by a Decision Making Process that produces an order that describe actions that cause effects that change the state into the desired End-State. The OIEM is to be a basis for a system design that combines the grammar and engineered knowledge with search mechanisms (Gustavsson et al., 2008).

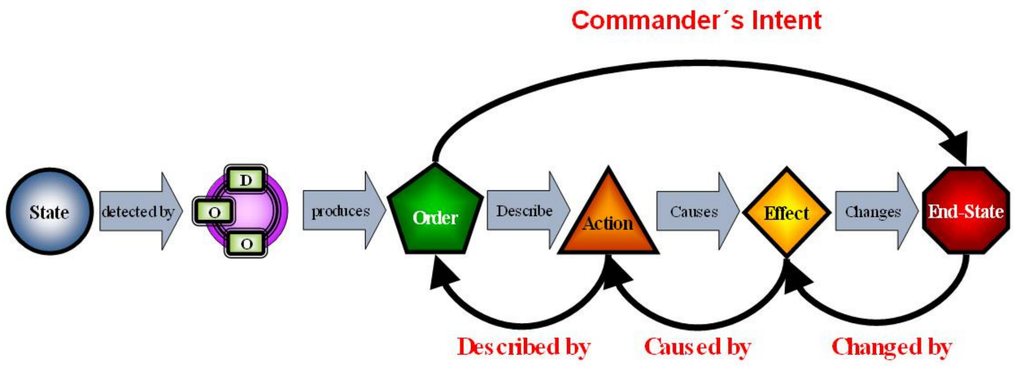


Figure 5. The Operations Intent and Effects Model where the OODA-loop's *Act* part is enhanced (Gustavsson et al., 2008).

[3] Gustavsson et al. (2008) refer to *expressives* as a component of CI that describes the *style* of the commander conducting the operations with respect to experience, risk willing, use of power and force, diplomacy, ethics, norms, creativity and unorthodox behavior.

The huge effort of developing a language capable of preserving the Commander's Intent to all participants of the command hierarchy is still in its prime, but will hopefully satisfy the ever increasing need to control, command, and coordinate the complex planning situations a coalition[4] may face.

2.2.6 Procedural tactics planning in a dynamic environment

Procedural combat tactics described by Straatman, van der Sterren & Beij (2005) evaluates the situation and terrain at hand by using on-the-fly algorithms and dynamic inputs. In a dynamic environment the use of dynamic procedural tactics is of the essence, as an AI using only static placed hints has trouble explicitly interpreting the abundance of input data (Straatman et al., 2005). Furthermore, the hints themselves might become compromised or illegal if the environment changes in an unexpected way, leading to absent or erroneous decisions. Baxter & Hepplewhite (2000) describe the future and to some extent the present state of the battlefield as "uncertain due to the possible destruction of other agents and potentially inaccurate sensor information", and states that planning efforts will be wasted if the they are due too far into the future with a complete set of actions.

By using procedural tactics the situation awareness is built on dynamic variables which more resemble a real world scenario, where for example *all* possible cover and fire positions aren't known, but rather *found* once a certain threat has been assessed at a certain position. This dynamic approach is however more likely to increase the CPU's computational need, as more calculations are needed in the absence of precompiled data. Another drawback according to Straatman et al. (2005) with the "procedural" nature of this approach, is that the AI is more complex to test and tune which leads to additional work constructing the environment to enforce the AI to behave in a certain way. Although, this can somewhat be administered easier with in-game debugging views like showing the agent's position picking and path-finding. As noted by Wallace (2004) the world state is constantly changing which does not allow a distinction to be made between planning and completing execution of a plan; it is valid only as long as its preconditions hold. Wallace (2004, pp. 229) identifies these problems with a reactive approach, which is present when using a regular FSM architecture without planning capabilities:

- The reactive approach relies on developers thinking of possible situations that might arise and how the agents should react to these situations. Planning eliminates this problem by introducing the ability for agents to solve problems themselves rather than having pre-canned solutions from the developer.
- The reactive approach makes it very difficult to deal with complex situations where the agent must perform a number of actions to achieve its goal.
- The reactive approach does not allow much scope for complex cooperative behavior between agents that can be tightly coordinated.

2.2.7 Goal-Oriented Action Planning

Goal-Oriented Action Planning stems from discussions from the GOAP working group of the A.I. Interface Standards Committee (Orkin, 2006), and has so far only

[4] "A set of heterogeneous entities including both military and civil governmental organizations as well as international and private ones, were not amenable to unity of command or a traditional hierarchy organized around strategic, operational , and tactical levels." (Alberts, 2007, pp. 4)

described the high-level concepts of GOAP. It is based on STRIPS (STanford Research Institute Problem Solver) which is an automated planner consisting of an initial state and goal states, which the planner is trying to reach by using actions that in turn have pre- and postconditions (Fikes & Nilsson, 1971). Long (2007) explains that the actions in GOAP stems from operators in STRIPS, and the primarily differences between the two techniques are the adaption of action costs in GOAP along with the use of the A* algorithm[5]. Much like the re-planning capabilities of the Goal-Driven Agent Behavior described by Buckland (2005, pp. 385), GOAP also incorporates a kind of re-planning feature, enabling the planner to construct a new plan consisting of other, but still valid, actions leading to the pursued goal world state (i.e. the end-state). The notion of using remembered (old) planning data and avoid planning from scratch, is called *Case-Based Planning* (Ontañón, Mishra, Sugandh & Ram, 2007), and could be incorporated in GOAP. The goals and actions available in a scenario using GOAP are static, i.e. no other goals and actions than those predetermined by the developer are ever used. Figure 6 depicts a planning example where the goal state `(kTargetIsDead, true)` can be reached by executing the actions `DrawWeapon`, `LoadWeapon` and `Attack`.

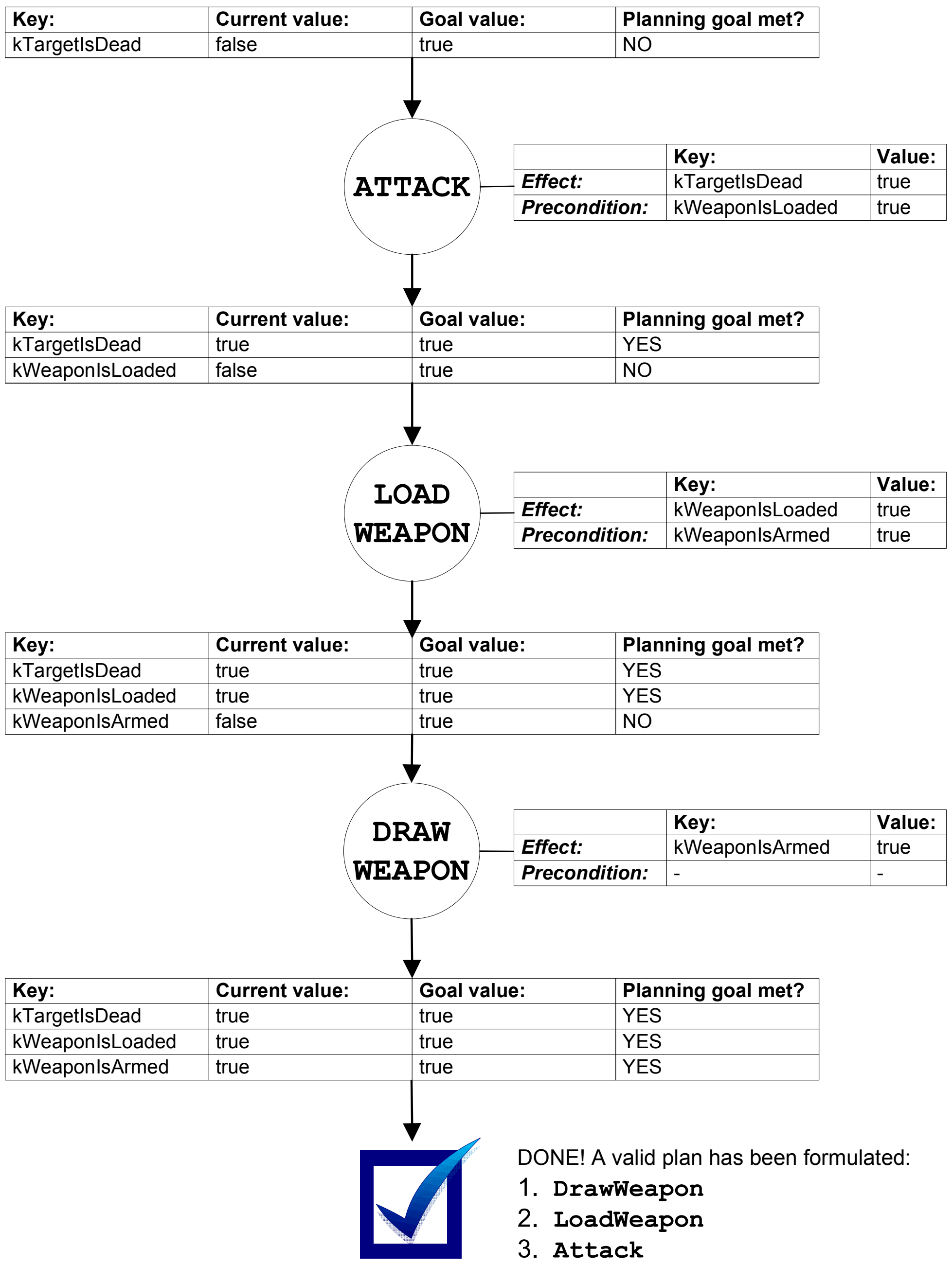
Orkin (2004c) suggests that the relevant world states are represented as a list of structures containing an enumerated attribute key, a value, and a handle to a subject. A specific action's or *operator's* preconditions and effects therefore has the same structure; both a precondition and an effect has a key and a value[6], e.g. one action called `GoToSleep` might have a precondition with the key `kBedlampIsOn` with the associated value `false`, and the effect might have the key `kIsTired` with the value `false`. An action can have multiple preconditions and effects, i.e. the action `GoToSleep` in the previous example may have additional preconditions and effects other than `(kBedlampIsOn, true)` and `(kIsTired, false)`. The actions are thus sequenced by satisfying each others preconditions and effects, and by doing so a plan pursuing the current goal (i.e. the desired end-state) is formed. Orkin (2004c, pp. 218) sees it as "each action knows when it is valid to run, and what it will do to the game world". It should be noted that an action only has the preconditions of interest to it.

Besides having a symbolic precondition or effect, an action may have a *context* precondition or effect, where the resulting value is dynamically obtained by an arbitrary function implemented by a certain subsystem. For example, the value of whether a threat is present could be directly retrieved from the world state `(kThreatIsPresent, true)` or from a context function `EvaluateThreatPresence` that returns a suitable value or handle. Long (2007, pp. 9) mentions two benefits of using context preconditions: "It improves efficiency by reducing the number of potential candidates when searching for viable actions during planning and it extends the functionality of the actions beyond simple Boolean world state symbols.".

In GOAP, each action has a static cost value which helps the planner to determine what actions are more favorable than others, i.e. the planner considers more specified actions before more general ones (Orkin, 2005). A* search is used to obtain the best path of valid actions which results in a plan. However, the use of static values has been criticized as they never change during the execution of the application (Long, 2007), making it harder to formalize a more expressive plan.

[5] The A* algorithm is described in detail by Buckland (2005, pp. 241).

[6] The value is either a integer, float, bool, handle, enum, or a reference to another symbol.



Key:	Current value:	Goal value:	Planning goal met?
kTargetIsDead	false	true	NO

	Key:	Value:
Effect:	kTargetIsDead	true
Precondition:	kWeaponIsLoaded	true

Key:	Current value:	Goal value:	Planning goal met?
kTargetIsDead	true	true	YES
kWeaponIsLoaded	false	true	NO

	Key:	Value:
Effect:	kWeaponIsLoaded	true
Precondition:	kWeaponIsArmed	true

Key:	Current value:	Goal value:	Planning goal met?
kTargetIsDead	true	true	YES
kWeaponIsLoaded	true	true	YES
kWeaponIsArmed	false	true	NO

	Key:	Value:
Effect:	kWeaponIsArmed	true
Precondition:	-	-

Key:	Current value:	Goal value:	Planning goal met?
kTargetIsDead	true	true	YES
kWeaponIsLoaded	true	true	YES
kWeaponIsArmed	true	true	YES

DONE! A valid plan has been formulated:

1. **`DrawWeapon`**
2. **`LoadWeapon`**
3. **`Attack`**

Figure 6. The planner finds a valid plan by a regressive search (after Orkin, 2004c, pp. 225).

For example, the action `SupportSuppressedSquadMember` probably should have a lower cost, i.e. be more favorable, if the supporting member is not under fire. These cost values resemble those mentioned by Reynolds (2002), who suggests using a priority system where different actions and circumstances are given different priorities. He furthermore states that "These [priority] values allow soldiers to be chosen by their current status, and for soldiers of a certain rank or health to be omitted unless their status is of a lower priority.".

Thanks to the general approach of GOAP it can be applied to virtually any problem scenario, while still having the benefits of a reasoning system that has an understanding of how goals and decisions relate, and what the effects might be

(Lagervik & Gustavsson, 2006). Here a few arbitrary examples are given where GOAP could be used as means of solving the problem at hand:

- **Assembling a chair**. By having the overall goal of assembling a chair, a plan is formulated where the first step might be to fit the legs *or* the armrests to the seating base. However, because mounting the legs prior to the armrests, the armrests are more easily fitted as the seating base is standing on its legs. Therefore, while both actions `FitLegs` and `FitArmrests` have no preconditions, `FitLegs` is given a lower cost value which enables the planner's search to favor the fitting of the legs prior to mounting the armrests. By rigging the legs and armrests, the world states `(kChairLegsIsMounted, false)` and `(kArmRestsAreMounted, false)` have changed to `(kChairLegsIsMounted, true)` and `(kArmRestsAreMounted, true)` due to the effects of the executed actions. The previous effects are in turn preconditions to the action `MountBack` which now can be executed, having the effect `(kChairIsAssembled, true)`. This enables further actions like `SellChair` or `ShipChair` depending on what actions are available and what actions the planner deems feasible.

- **Baking a pie**. If the overall goal is to bake a pie the end-state could be `(PieIsBaked, true)`. The plan is then formulated based on the actions available and their costs, which for instance might include `TurnOnOwen`, `PutBowlOnTable`, `AddEggToBowl`, `AddButterToBowl`, `AddFlourToBowl`, `MixBowlIngredients`, `PutBowlInOwen`, `Wait`, and `PutBowlOnTable`. In this case the first action would be `TurnOnOwen` which has no preconditions, just as `PutBowlOnTable`. However the actions `AddEggToBowl`, `AddButterToBowl` and `AddFlourToBowl` might have the precondition `(BowlIsOnTable, true)` which is the very effect of the action `PutBowlOnTable`. If the mixture is supposed to fare better if the egg is used before that butter, which in turn is used before the flour; then `AddEggToBowl` is given the lowest cost value of the three actions whilst `AddButter` is given a lower cost value than `AddFlourToBowl`. The baking is then allowed to continue by utilizing the remaining actions of the plan, resulting in a baked pie once the final action `PutBowlOnTable` has been executed.

- **Telling a story**. If the overall goal is to reach the end-state `(StoryHasBeenTold, true)`, various subparts of the story may be linked by preconditions and effects which enables a dynamic story based on the variables at hand. For example, the story could start by reading a light sensor which acts as a precondition for one of the introduction subparts. After the introduction has been experienced, the user's pulse is checked which may act as a context precondition of some other subpart of the story, which may lead to less creepy story parts being told until the user is sufficiently calm.

- **Using a tutorial**. Much like the story telling example, a tutorial might be modeled using GOAP, allowing it to display hints according to the user's contextual preferences or other aspects of the scenario.

As seen from the examples above, there are many areas of application but GOAP has so far, what is known, not been particularly used in these novel areas. The reason for this is probably that the technique is new and has not yet been fully accepted, realized or understood by the market. Few official implementations of GOAP exists, and while there are some high-level guidelines, the very founders of GOAP have not yet created an interface standard. The GOAP architecture should thus be seen more as a guideline than a de facto standard. Because of this it seems that components incorporated in GOAP could be excluded as well as others being included if it serves the overall implementation.

An agent using GOAP has a *working memory* in which the agent's SA is updated and stored for later use in conjunction with a blackboard. The role of a blackboard in GOAP is to act as an intermediate information repository between various subsystems, which enable them to share data without having direct references to each other. Because certain agent sensors may have expensive computational costs, the computations are made over many frames and results are cached in the working memory (Orkin, 2005). The working memory's cached data is stored in the form of `WorkingMemoryFacts` which are divided into different types depending on the scenario at hand and coupled with the experienced confidence of the fact. A baking scenario might have a `Cupcake`, `DanishPastry`, `Event` and `Smell` as facts, whilst the combat scenario created by Long (2007) included facts like `Enemy`, `Friendly`, `PickupInfo` and `Event`. By employing these kinds of facts the planner is able to query the working memory in the best manner it may seem fit, e.g. by querying what `Cupcake` smell *most*. As GOAP depends on situation awareness, parts of it can be modeled by the OODA-loop or activity and organizationally as done by Lagervik & Gustavsson (2006). Orkin (2005) states that the GOAP architecture resembles that of the MIT Media Lab's C4 model depicted in Figure 7, the main differences being additional subsystems and a planner as action system.

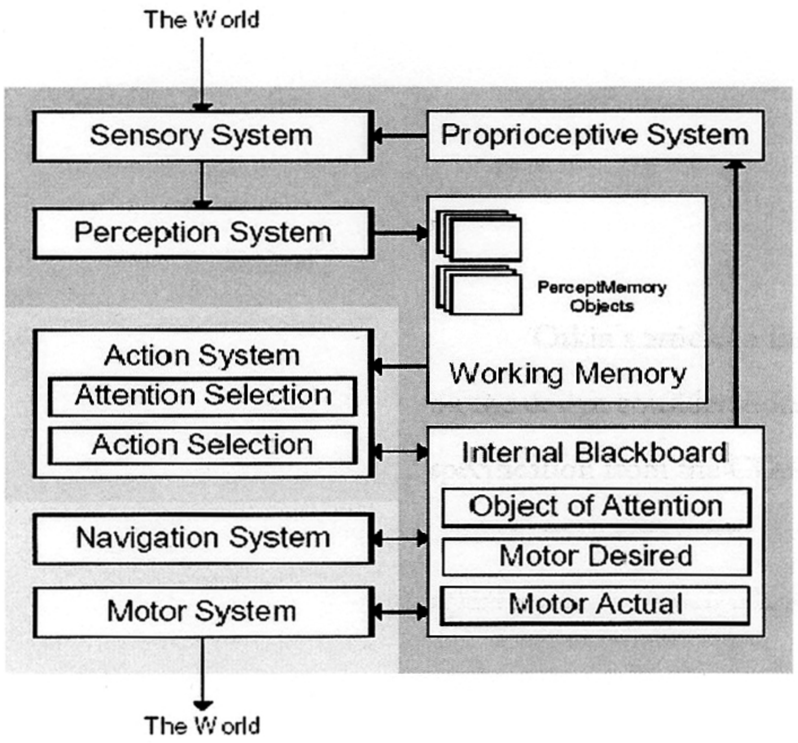


Figure 7. MIT Media Lab's C4 architecture (Burke, Isla, Downie, Ivanov & Blumberg, 2001).

Orkin (2006) speaks of the three benefits of planning being decoupling of goals and actions, layering behaviors, and dynamic problem solving. He also states that FSMs are procedural while planning is declarative, and highlights how a FSM exactly tells an AI how to behave in every situation; whilst a planning system relies on the AI finding its own sequence of actions that satisfy the pursued goal. Even though GOAP does not include any explicit squad behavior, it can be employed by imposing a cost factor to a certain goal and thereby provoke a desire for an agent to strive towards that goal. A strong advantage of GOAP identified by Orkin (2004a, pp. 1) is that "atomic goals and actions of a GOAP system are easy to read and maintain, and can be sequenced and layered to create complex behaviors"; the gain of simplicity and modularization makes it easier for multiple developers to collaboratively implement complex behaviors. Layered behavior is accomplished by inheritance where subsequent behaviors or actions implement the same basic behavior as their parents. For example may an `Attack` action serve as an abstract parent to more specified children attack actions like `SneakAttack` and `ThrustAttack`.

By using GOAP, agents may make decisions not anticipated by the developer as the chain of actions (the plan) is made up at runtime (to the developer's joy or dismay). GOAP incorporates the use of finite state machines, but they are said not be as hardly coupled to the transition logic as an original FSM model. Instead of having the states themselves decide the transition, the various actions handle the transition logic (Orkin, 2004c). As stated by the Working Group on Goal-Oriented Action Planning (2004), the overall design of GOAP can be divided into three separate systems:

1. A goal selection interface that weighs up relevance, probabilities, and so on, and decides which goal to plan towards.
2. A planner that takes a goal and plans a series of actions to achieve it.
3. An action interface for carrying out, interrupting, and timing actions that characters can carry out.

Figure 1 depicts these subsystems as the goal selection interface is incorporated into the style layer that selects an appropriate goal world state depending on the personality of the agent. The planner that takes a goal and plans a series of actions to achieve it is depicted as a reasoning brain. The action interface is comprised in the embodiment of the agent or agents performing the suggested action, leading to the desired world state transition.

The Working Group on Goal-Oriented Action Planning (2003) recognize GOAP to have the ability to provide more flexible and diverse behaviors than FSMs and rule based systems, since plans are constructed "online" from atomic actions and adapted specifically to the current goal. Instead of a simple finite state machine reactively directing an agent in every situation, GOAP is employed to formulate a plan that strives to realize a certain goal world state, that in turn has been found by a goal selector. This goal selector may be an internal implementation as a set of functions, each representing a certain goal which depending on the scenario returns a goal relevancy value. The goal that returns the highest relevance value is pursued; however a cut-off implementation may stop the most promising goal from being pursued if its relevance value is too low, leaving the agent idle until a sufficient relevance value is obtained. Another implementation aspect is to externalize the goal selector completely, having the GOAP planner receiving an order containing the desired goal world state. Choosing goals by a relevancy value is mentioned by O'Brien (2002, pp. 379):

> In its simplest form, goal prioritization could be merely ordering the goals on the basis of the score assigned to them (…) This is a perfectly valid approach, and the only other thing that might be required is to have a maximum number of dynamic goals that are allowed to remain on the list.

The pool of available actions for a certain agent may differ from that of another agent; but both agents are theoretically able to accomplish the same goal, given that there exists a valid action sequence leading to the pursued goal. This also means that an agent can be forced to use only a subset of its valid actions, in order to simulate a certain strategy or personality, i.e. style.

2.2.8 Regressive searches

GOAP employs real-time planning by using regressive searches in order to find appropriate actions that satisfy the pursued goal. The plan formulating algorithm starts its search at the goal and then works its way towards the current state, by finding actions with the appropriate preconditions and effects. The search does not find all possible plans, but instead finds a valid plan that, thanks to the A* algorithm, is the most promising, i.e. cheapest plan. Because of the regressive search, GOAP can not deliver a valid incomplete plan as it does not contain the initial action step leading from the current state to the next, hence it is not an anytime algorithm[7]. For example, if the planning system was intercepted at stage 2, as illustrated by Figure 8, the plan would only contain the action sequence leading from a certain state (F or G) to the goal state (H); it would not contain the first necessary action step leading from the current state to the next (i.e. from A to B, C, D or E).

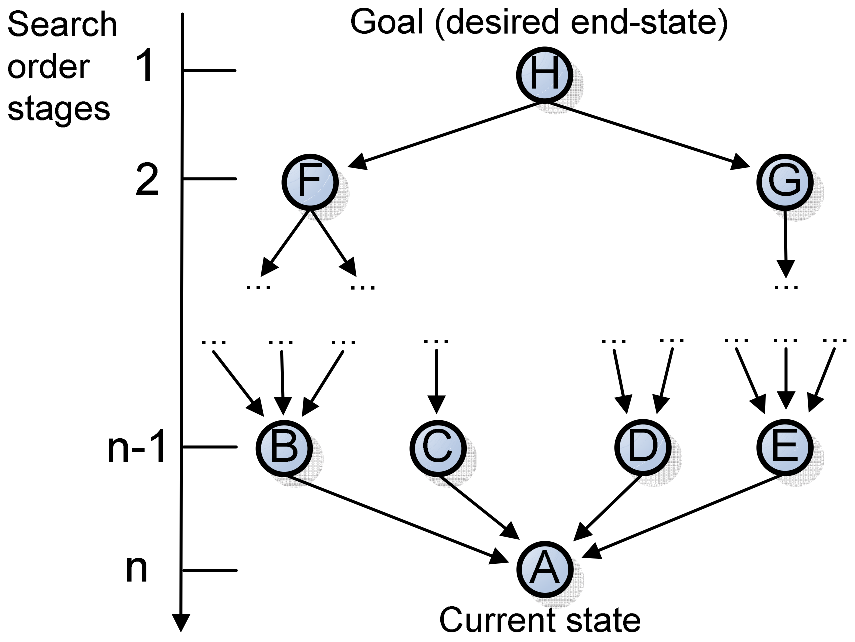


Figure 8. The plan formulation start at the goal state H, working towards the current state A by finding appropriate actions.

[7] An *anytime algorithm* can be aborted and still deliver a valid result, even though it did not finish.

2.2.9 Effect based planning

Effect based planning (EBP) aims to clarify the effects in different areas regarding the actions incorporated into a plan. By utilizing the EBP concept, negative consequences are to be foreseen and avoided or worked against in order to achieve a desired goal with as few side-effects, i.e. unintended effects, as possible. EBP has a strong connection to Commander's Intent since the planning entities must fully understand (or share) the CI in order to formulate a plan with actions causing the desired effects according to the CI. As stated by Farrell (2004), CI is central to EBP. FOI, the Swedish Defense Research Agency, has through Schubert, Wallén & Walter (2007) presented an EBP technique using a cross impact matrix (CIM) which "…is to find inconsistencies in plans developed within the effect based planning process." (Schubert et al., 2007, pp. 1) using Dempster-Shafer theory and sensitive analysis. The CIM consists of all *activities* (A), *supporting effects* (SE), *decisive conditions* (DC) and *military end state* (MES) of the plan, and should be seen as a dynamic entity built and continually managed by a broad working group with a strong knowhow of the various matrix components and their interaction (Schubert et al., 2007). By employing this CIM architecture, any weaknesses and all strengths of the plan can be found prior to the effect based execution phase, giving the involved participants a more similar understanding of the situation leading to better decisions being made. The actual plan formulated through EBP is described by Schubert et al. (2007) as a tree structure having the MES as root. Figure 9 depicts a plan formulated according to EBP.

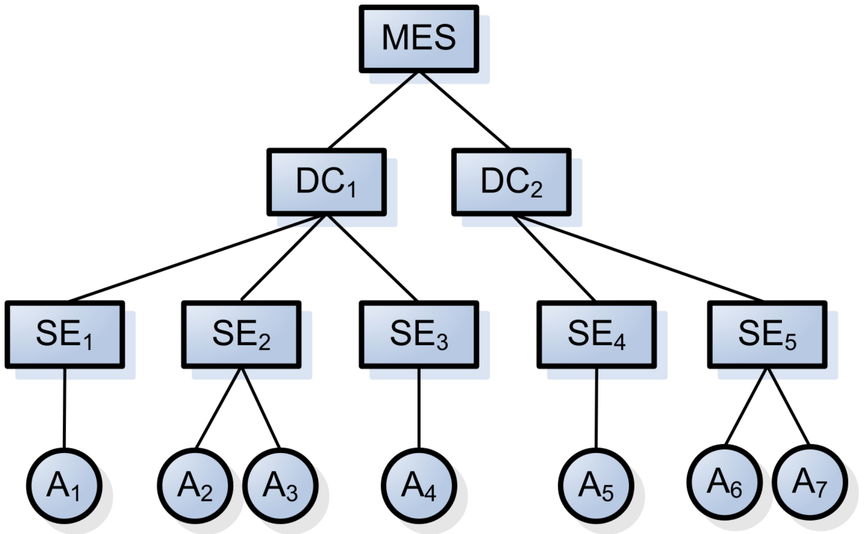


Figure 9. The plan formed according to EBP (after Schubert et al., 2007, pp. 2).

Comparing this structure in this context to others, one can imagine the similarities of a composite/atomic goal breakdown by Buckland (2005), or composite/simple task and action described by Dybsand (2004). In that sense DC could be seen as a composite goal or task, while SE maps to a composite goal or simple task[8], and A maps to an atomic goal or action. Regardless of the naming convention, MES is without doubt reached by the occurrence of A, SE and DC.

[8] *Composite* and *simple* tasks are sometimes also referred to as *compound* and *primitive* tasks (Erol, Hendler & Nau, 1994; Hoang, Lee-Urban & Muñoz-Avila, 2005).

As effects from actions are modeled in GOAP, the planning taking place could somewhat be seen as EBP since the known effects may judge what actions get incorporated into the plan.

2.2.10 Hierarchical Task Network

A Hierarchical Task Network (HTN) is a set of tasks ordered in a hierarchical fashion where the root task corresponds to the overall completion of *the* task, i.e. the utmost goal. The plan formulation can be thought of as a funnel where an abstract unexecutable high-level plan is first formulated (Wallace, 2004) in order to formulate more precise and ad hoc plans executable in the current situation. Just as in STRIPS-planning, the *operators* are the effects of a task. Each task is either *primitive* or *non-primitive* (i.e. *compound*) where a non-primitive task contains other tasks, which in turn could be either primitive or non-primitive (Erol et al., 1994). A non-primitive task can thus not be performed directly which is the case with a primitive task. In order to solve a non-primitive task, a *method* is applied which *reduces* the task, forming a new task network consisting of primitive and/or non-primitive tasks (Wallace, 2004). Muñoz-Avila & Hoang (2006, pp. 419) explains HTN planning as high level tasks being decomposed "…into simpler ones, until eventually all tasks have been decomposed into actions.".

According to Erol et al. (1994, pp. 1124) "The fundamental difference between STRIPS-style planning and HTN planning is the representation of 'desired change' in the world.". These "desired changes" are represented in the HTN as compound tasks which both incarnate a goal and evolves to contain the necessary actions to reach it. If a HTN task is seen and handled as an independent goal, then that goal could be fed to a GOAP planner which would work out a plan of more low level action details, whilst the HTN keeps track of the overall goal and subgoals. According to Wallace (2004) one of the most powerful features of HTN planning is its ability to perform *partial re-planning*, which in contrast to a regressive planner in GOAP does not have to formulate a whole new plan if the plan should fail. Muñoz-Avila & Hoang (2006) sees the reasoning process as the most crucial difference between STRIPS and HTN, where STRIPS reasons in level of actions whilst HTN reasons in level of tasks. Figure 10 depicts a HTN build up where the primitive tasks (i.e. the actions) are incorporated in a subtask, which in turn builds up all the way to the top-level task.

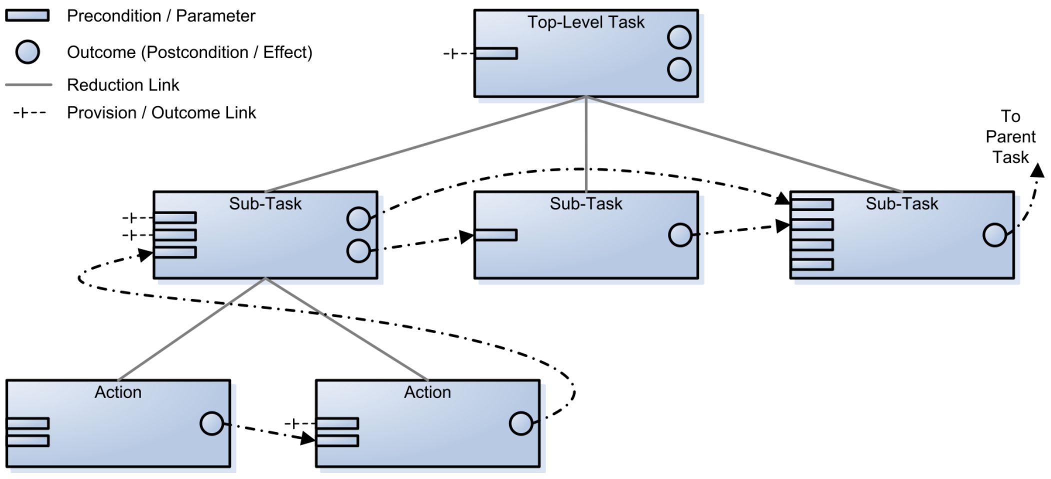


Figure 10. A hierarchical task network (after Paolucci et al., 2000, pp. 4).

2.3 Multi-agent and squad behavior

According to van der Sterren (2002a) the most appropriate action to perform is determined by each AI member taking account of the situation of his teammates and the threats known to the team, in addition to his own state. This is in the area of situation awareness which must be considered as a main precursor to decision-making, but it is nonetheless an independent process that must be recognized separate from the decision-making (Breton & Rousseau, 2005). The communication, interpretation and overall understanding of the intent between agents, determines whether a particular task succeeds; and there exists a need for a standard to communicate intent in a simulation of complex planning (Borgers et al., 2008). The complexity of a scenario incorporating plans for numerous agents can rapidly grow, not least from the resource handling stemming from the parallel tasks taking place. In a multi-agent environment where high-level strategic goals need to be dealt with, Muñoz-Avila & Hoang (2006, pp. 419) claims that even though it is possible to encode these strategies in STRIPS representation, a HTN is preferred since it "capture strategies naturally because of the explicit representation of stratified interrelations between tasks". The STRIPS-HTN relation could thus exist in symbiosis as the HTN planner reasons on what strategy to abide whilst the GOAP planner (i.e. a STRIPS realization) handles the more fast paced low-level action decisions.

In the SAMPLE (Situation Awareness Model for Pilot-in-the-Loop Evaluation) architecture, cognitive processing at squad commander level is made up of two streams consisting of tactical assessment and terrain assessment, which are responsible of rating the combat effectiveness of the squad (Aykroyd, Harper, Middleton & Hennon, 2002). By utilizing a *Battle Assessment Matrix* working with a list of known enemies and squad info, factors such as amount of team members and weapons used may determine the fuzzy rating of the squad's combat effectiveness (Aykroyd et al., 2002). The result is then together with the terrain data evaluated by expert systems and belief networks in order to determine the squad's activity for the current decision cycle. Individual agent combatants in SAMPLE receive various orders, where for example a combatant agent might be told to fire at a certain enemy or to restrict its movement to an area within the squad's line (Aykroyd et al., 2002). The combatants actual decisions stems from numerous analyzers where for example the *Shot Analyzer* can point out firing positions as well as weapon choices considering a certain enemy. Depending on what orders have been given, a combatant's selected threat target and best shot may be overridden as long as the target isn't deemed a critical threat, which for instance could be a nearby enemy with a clear shot and an assault rifle (Aykroyd et al., 2002).

According to Paolucci et al. (2000) it is of the utmost importance that planning systems in a multi-agent environment allow *execution while planning*, in order to allow "flexible interaction with other agents or with the domain while the agent plans locally" (Paolucci et al., 2000, pp. 18). In a multi-agent environment where GOAP is employed, various commands or doctrinal rules and conditions may be considered by inflicting a value or "cost" change of an agent's goals and/or actions, making the goal or action more or less favorable. Doris & Silvia (2007) states that "A GOAP system imposes a modular architecture that facilitates sharing behaviors among agents and even across software projects.". Depending on the cost degree and the agent's susceptibility of the change, an agent might do the bidding of its surroundings. For example, a popular commander giving a dangerous move order to a military agent, might due to its expressives[9] considerably lower the cost of the `Move` action, making other actions like `StayInCover` or `Flee` less favorable. Hence, the military agent chooses and executes the `Move` action, even though the action per se is less favorable according to the agent. Another example is a doctrine stating that women and children shall be saved prior to men in a fire emergency. This doctrine could be implemented by giving the goals or actions `SaveWomanFromFire` and `SaveChildFromFire` a higher relevance value than `SaveManFromFire`. The commands, doctrines, personality preferences or other implicit conditions as described in 2.2.5, are thus applied as filters or layers on the primal functions. As more of these layers are applied a complex and implicit group behavior may emerge which takes many affectors into consideration. The GOAP architecture, much like the SAMPLE architecture, is thus capable of reacting to an adversary in a dynamic environment while employing a highly modular and extendable system.

2.3.1 Centralized vs. decentralized approach

A squad may be decentralized meaning that there exists no obvious squad leader; all squad members have the ability to issue suggestions based on their perceptions and person preferences, i.e. style. Van der Sterren (2002a, pp. 235) identifies the following attractive reasons of choosing a decentralized approach:

- It simply is an extension of the individual AI.
- It robustly handles many situations.
- It deals well with a variety of capabilities within the team.
- It can easily be combined with scripted squad member actions.

In order to optimize the squad's situation awareness, it is most important that the squad members communicate and share their intentions and observations. By doing this the squad may somewhat become a single entity with each member acting more as a sensor or suggestion input than a free roaming entity. Each squad member may thus be seen as a tool or weapon with which the entity, i.e. the squad, can achieve its goal; much like a human hand attains an arbitrary goal by using its squad members – the fingers. However, the decentralized approach to squad AI is just a solid base from which to work, and it is easily extended with increasingly realistic tactics (van der Sterren, 2002a).

In a centralized squad the communication and orders are sent in a hierarchical manner, possible of having multiple echelons. The benefits of this approach are according to van der Sterren (2002b):

- The amount of problems each type of AI has to cope with is significantly reduced.
- Each kind of AI can be changed and specialized independently from each other.
- Lower CPU demand as a result of fewer computations.

The main problems when dealing with a centralized approach are those of conflicting objectives. The AI of a squad member is usually not able to see the mission's grand scheme and may have different goals than those of the squad commander's AI, e.g.

[9] *Expressives* as described in 2.2.5 (Gustavsson et al., 2008).

the squad member might prefer to pick up a better weapon instead of directly dealing with a certain threat seen by the squad commander (van der Sterren, 2002b). To deal with these conflicts van der Sterren points out that the squad members must know the rules-of-engagement based on the current situation. For example a situation where the squad's actions must be coordinated meticulously is more sensitive to ego based behavior. Van der Sterren (2002b, pp. 249) further claims that "while there is no such thing as the optimal command style, you will go a long way addressing your squad requirements by a smart mix of the two styles [decentralized and centralized approach]…".

In a dynamic real world scenario it is often desirable to avoid detailed low level command and control (C2)[10], and as stated by Alberts (2007, pp. 1) "The future of command and control is not *Command and Control*. In fact, the term *Command and Control* has become a significant impediment to progress.". It is in most cases instead preferable to only express orders (goals) in high level terms, i.e. intent, which the lower level echelons (agents) then may solve how they may see best fit, and by doing this exchanging *command and control* in favor of *focus and convergence* (Alberts, 2007). This means that even though there exists a much needed command hierarchy, the commander should not act in a strong centralized manner and thereby hide relevant information and pinpoint actions to the subordinates. Instead, he or she should allow the subordinates to *self-synchronize*[11] and be part of *Network Centric Warfare* (Smith, 2003; Alberts, 2007). This relatively modern operational doctrinal view offers the subordinates a broader view of the situation as they together may decide on what course of actions to take, much like using a blackboard (see section 2.2.2) and emphasize emergent behavior, in contrast to being micro managed by a high level commander that perhaps has inadequate situation awareness. Alberts (2007, pp. 1) mentions the three core concepts agility, focus and convergence; where "…agility is the critical capability that organizations need to meet the challenges of complexity and uncertainty; focus provides the context and defines the purposes of the endeavor; convergence is the goal-seeking process that guides actions and effects."

GOAP do employ both a decentralized and centralized approach in that a specific order is having the effect of a certain goal having a higher preference, thus allowing the squad member to consider the commander's will, whilst still, to some extent, maintaining its own prioritization of goals. A squad in GOAP is therefore not strongly coupled to its commander which also allows for easy substitution in case the commander meets his or her demise or is otherwise compromised.

[10] C2 is defined by Department of Defense Dictionary of Military and Associated Terms (Joint Publication 1-02, 2001) as: "Command and Control: The exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of the mission. Command and control functions are performed through an arrangement of personnel, equipment, communications, facilities, and procedures employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of the mission…".

[11] "Self-synchronization is a way of distributing decision rights and is associated with a specific set of patterns of interactions among participants." (Alberts, 2007, pp. 10)

3 Problem

The purpose of this project is to extend the GOAP architecture with threat analysis capabilities by slightly modifying the GOAP architecture. This would enable entities using GOAP to have increased situation awareness as they are capable of determining how big a threat an assessed entity poses. By utilizing this knowledge and make the assessing entity communicate its findings to other entities (both virtual and real), perhaps better strategies and decisions could emerge. The inherent generic benefits of the GOAP architecture should also comprise the threat analyzer as it would perform planning in a similar way. Hence the threat analyzer could be seen as a GOAP planner, but instead of delivering an action plan the threat analyzer is to determine how big a threat a certain entity constitutes, based on that entity's current world states and what actions are known to the threat analyzer.

Much like how an ordinary GOAP planner is loaded with a set of actions, the threat analyzer is to be loaded with those actions it should be able to account for when performing a threat analysis; enabling it to formulate a "risk plan" given a certain goal world state fed by an arbitrary input, like a doctrine or a command. The goal world state of the analyzer is thus more as an "anti goal" or *unwanted* world state, and the world states deemed dangerous will be substeps of a plan leading to the unwanted world state. By comparing an entity's world states by those found in the formulated risk plan, the analyzer should be able to determine the threat level and its completeness of the entity being assessed. In other words, the analyzer is to somewhat anticipate the occurrence imminence[12] of its anti goal world state.

The experiment will be seen as a success if the implemented threat analyzer is able to assess a certain entity and assign it a corresponding threat level and completeness, by using a simplified GOAP architecture[13]. The benefits of this threat analyzer in conjunction with GOAP should among other things be the simplicity of not having to flag a multitude of individual world states as "dangerous", but rather just flag one or a few world states as undesired; from which all relevant interconnected dangerous world states, i.e. threats, can be found when assessing a certain entity.

- **Objective 1: Development**. Develop an architecture consisting of components relevant for the evaluation of the threat analyzer using GOAP, i.e.
    - A goal selection interface that choose the most favorable goal which then can be planned against.
    - A GOAP planner that takes a goal world state and formulates a plan consisting of actions to achieve the goal.
    - A threat analyzer that takes an "anti goal" world state and evaluates the threat level of a certain entity's current world states.

[12] The closer an entity is to achieving an undesired world state (from the threat analyzer's point of view), the bigger threat it poses.

[13] The need of utilizing the complete GOAP architecture with all performance optimizations is most likely an overkill, since the scenario will hold only two entities with a small amount of actions.

    - An (agent) entity having a seeing sense capable of spotting and setting world states, as well as an action interface for carrying out, interrupting, and timing actions.
    - A graphical representation of the world and entities performing actions and assessment.

- **Objective 2: Evaluation**. Evaluate the developed architecture and identify its shortcomings and strengths considering its inherited GOAP properties, implementation complications, usability and flexibility.

4 Methods

This chapter discusses the methods at hand and why a certain method was chosen, and how it is supposed to be carried out.

4.1 Method for objective 1: Development of the necessary components

As the problem phenomena implicates that development will be in full control and since other Goal-Oriented Action Planning (GOAP) implementations do not have a threat analyzer as proposed in chapter 3, a case study will not be suitable. Furthermore, as in most AI research, a lot of knowledge is gained by *trying out* new concepts by building a hands-on implementation where theoretical assumptions may be deemed correct or false. In this case it is assumed that GOAP could serve as a foundation for a threat analyzer, and it would be possible to perform interviews or surveys about the likeliness of a threat analyzer based on GOAP being successful or not, and how it should be built. However, it seems sufficient in regard to the complexity of this project to use the *implementation* method as it concisely will show a hands-on realization that consider available information[14] about the GOAP architecture.

4.2 Method for objective 2: Evaluation of gathered data and observed behavior

The gathered data and observed behavior could be analyzed in the light of an interview or a survey, but these methods are, in this case, inferior to a swift experiment taking place in the built implementation. By employing a tailor-made experiment to the implementation, an evaluation of the built architecture can easily take place, which fulfils the needs of the evaluation. So even though some interesting comments probably could be gathered by using an interview or a survey, the method chosen for objective 2 is *experiment*, as it sufficiently covers the problematic aspects of objective 2.

The graphical representation in combination with threat data logs from the threat analyzer should make sure that the evaluation stays consistent and answer the questions stated in objective 2. For example, it shall be possible to graphically observe how an entity performs a certain action and cause a world state change, whereby the threat analyzer may assess the entity and tag it with its determined threat level.

It would also be possible to employ the implementation method in order to build some kind of data evaluation application. This is however unnecessary in this case since the amount of logged data will be relatively small, permitting it to be displayed by other means.

[14] The GOAP architecture will primarily be assessed through the various articles by Jeff Orkin, the AIISC reports, the work done by Long (2007), and the F.E.A.R. SDK.

5 Realization

This chapter describes the realization of the project's implementations. Section 5.1 presents an overview of the realization where key subjects and framework decisions are explained. Section 5.2 and 5.3 describes the project's identified actions, goals and world state keys as well as certain design decisions. Section 5.4 gives a more profound picture of the implementation and shows various key functions in pseudocode. Finally, section 5.5 states the scenario's rules and conditions while section 5.6 depicts the project's class diagram.

5.1 Overview

Because earlier implementations of Goal-Oriented Action Planning (GOAP) have used C++ the same language will be used in this project. By doing this the architectural consistency should be intact as no obvious gain can be seen in using a new language per se. To achieve easy and accessible results, the scenario will use simple 2D graphics through the C++ API *Simple and Fast Multimedia Library* (2008). In order to keep focus on the relevant issues and not letting the project get out of scope, performance optimizations and parameters not critical for the evaluation of the threat analyzer will be left out[15]. To evaluate the threat analyzer, a scenario will be set up where two competing entities strive towards their opposite goals using a simple GOAP planner to formulate an action plan. The competitors will be equipped with a seeing sense, able to identify agents and items residing in a walled rectangular 2D world, as depicted in Figure 11.

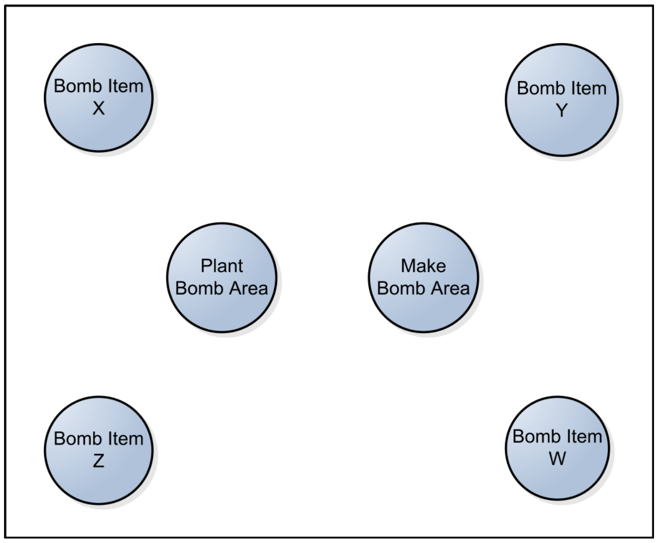


Figure 11. The simulated world where six key areas reside.

One of the competitors, the *Bandit*, will have the goal of reaching the world state `(kTargetIsBlownUp, true)`, which will be done by amassing four bomb items by walking up to them. Once the items have been collected, the Bandit will be able to make and plant a bomb at the make bomb and plant bomb area respectively. Upon

[15] A* searches and action costs are not to be implemented since they are deemed unnecessary, as this implementation will only have a small amount of actions. As long as an action is valid, it will be chosen stochastically depending on its position in the list of available actions.

planting the bomb, the value of the goal world state key `kTargetIsBlownUp` will change from `false` to `true`, meaning that the Bandit has reached its goal. The other competitor, referred to as the *Law Enforcing Coalition Force* (LECOF), will have a threat analyzer, loaded with the same goal world state as the Bandit, i.e. `(kTargetIsBlownUp, true)`. When the LECOF's seeing sense spots the Bandit, the LECOF's threat analyzer is to assess the Bandit's world states according to Figure 12.

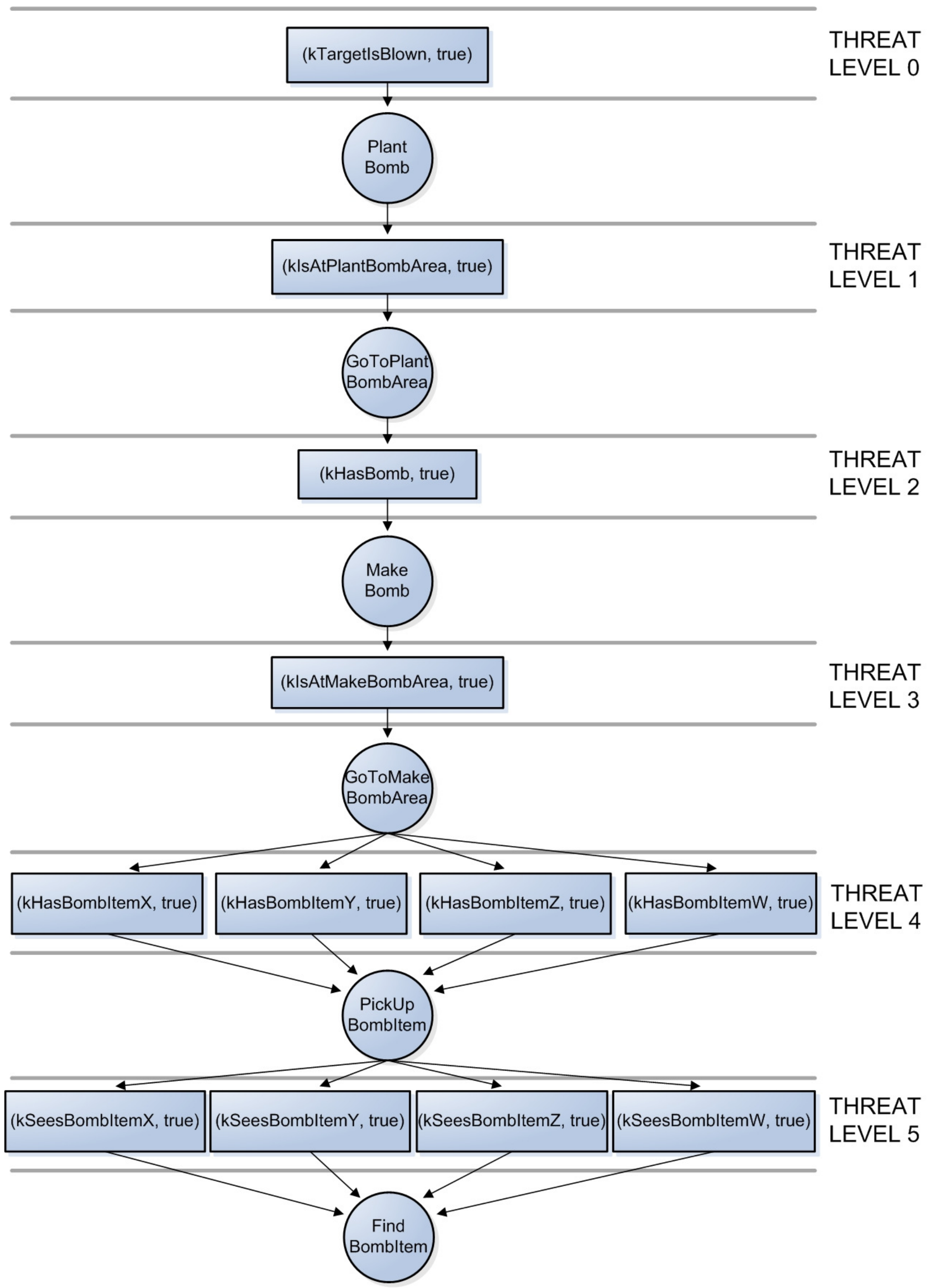


Figure 12. The action graph used by the Bandit's GOAP planner and the LECOF's threat analyzer (rectangles represent world states, circles represent actions).

The action graphs of the Bandit and the threat analyzer will thus be identical, but used in different ways. The actions (circles), as depicted in Figure 12, are loaded into the Bandit's GOAP planner and the LECOF's threat analyzer, which is to assign the Bandit a certain threat level and completion percentage depending on the world states assessed. If, for example, the LECOF spots a Bandit that is carrying bomb item x and has all world state keys set to `false` (except `kHasBombItemX`), the threat analyzer should be able to find out that the Bandit is of threat level 4 with a 25 % completion, i.e. one of four of that level's world states correspond to the threat analyzer's action graph.

The following components have been identified and deemed necessary for the project:

- **Entities, i.e. agents and bomb items**. The embodiment of an agent or bomb item represented by a graphic primitive or picture. The simulation will contain two competing agents, and the bomb items will be used as means of changing the world state (and consequently the threat level) of the Bandit agent.
- **Senses**. Each agent will have a seeing sense responsible for detecting other agents and bomb items. The field of view will be conical and the see range will be limited in order to have the agents look for each other or bomb items.
- **Working memory**. Each agent will have its own working memory which will contain representations of sensed entities of the world, i.e. memory facts. For instance may the seeing sense post its primitive findings to the working memory, where they later on may be accessed, questioned or evaluated by other subsystems of the agent. This does in some sense let the working memory have the role of a blackboard, that for example could be asked what known entity is currently closest.
- **Simple GOAP planner**. The simplified GOAP planner will be responsible of formulating a plan given a certain goal world state, found by the goal planner. As performance optimizations will not be necessary given the size of this project, A* searches will be replaced by brute force searches[16].
- **Threat analyzer**. The threat analyzer will be responsible of calculating the threat level and its completeness of a certain entity by assessing the world states found in that entity, given a certain unwanted world state, i.e. "anti goal" of the threat analyzer.
- **Plan**. The plan represents a component that holds a set of necessary actions which will enable its user to reach its current goal world state. The plan will be passed from the GOAP planner to its owner, and is to be replaced with a new plan whenever the plan fails or completes.
- **Goal planner**. The goal planner will be responsible of finding the most promising goal, i.e. the goal with highest relevancy. Once the most promising goal has been found, its goal world state is returned and used by the GOAP planner in order to find a suitable action plan.[17]

[16] A *brute force search* could also be described as an *exhaustive search* where in this case *all* actions are examined in order to formulate a plan.

[17] The goal planner could be seen as a commander telling or influencing the agent (i.e. the agent's GOAP planner) what the desired effects or end-state is. The commander is thus allowing the (subordinate) agent to formulate a plan of its own, i.e. to find a set of actions, not specified by the

- **Goals**. Goals will be loaded into a goal planner and consider the current world states and/or other contextual data in order to calculate their relevancy.
- **Actions**. Actions are representations of world state transitions which are linked by preconditions and effects. Actions will be used by a GOAP planner or threat analyzer in order to formulate a plan or to find dangerous (unwanted) world states.

5.2 Identified actions and goals

To keep the implementation simple and focus on threat analysis with GOAP, only two goals will be implemented, one for each agent. This should ease the debugging complexity as well as keeping the scope of the scenario to a manageable and intuitive size, regarding the needed actions. One could of course argue about the need of a goal planner when only one goal exists, but the architecture will this way be extendible if a need of more goals arises. So even though there will exist only one goal per agent, the functionality of the goals will be uncompromised, and each goal will have a `Relevance` function responsible of returning a representative value of the particular goal to the goal planner.

In order to build a scenario of the suggested design the following goals have been identified and need to be implemented:

- **Plant bomb**. The goal `PlantBomb` will be pursued by the Bandit agent and has the goal world state `(kTargetIsBlownUp, true)`. This goal will also be loaded into the LECOF's threat analyzer.
- **Find threat**. The goal `FindThreat` will be pursued by the LECOF agent and has the goal world state `(kSeesThreat, true)`.

To enable the agents to reach these goals and make an interesting scenario, the following actions will also have to be implemented:

- **Find Bandit**. The action `FindBandit` will enable a LECOF agent to find a Bandit by roaming the world in a random direction until it hits a wall or sees a Bandit. If the LECOF hits a wall a new direction is randomly chosen. This means that no neat steering behavior is used since it is not required for the problem at hand.

  Precondition: None

  Effect: `(kSeesBandit, true)`
- **Assess Bandit**. By using the action `AssessBandit` a LECOF may assess a seen Bandit (the Bandit's world states). Depending on the Bandit's world states the LECOF's threat analyzer comes up with the current threat level and completion of the Bandit assessed.

  Precondition: `(kSeesBandit, true)`

  Effect: `(kSeesThreat, true)`
- **Find bomb item**. The action `FindBombItem` will work much like the `FindBandit` action and will enable the Bandit agent to find a bomb item. A Bandit using this action will roam the world in a random direction until it hits a wall or sees a bomb

commander. Also, note that the goal planner could be an incarnation of a HTN planner that handles the overall strategy and feeds world state goals to the GOAP planner.

item. If the Bandit hits a wall a new direction is randomly chosen. Just like the LECOF agent, the Bandit does not use any advanced steering behavior.

Precondition: None

Effects: `(kSeesBombItemX, true) OR (kSeesBombItemY, true) OR (kSeesBombItemZ, true) OR (kSeesBombItemW, true)`

- **Pick up bomb item**. The action `PickUpBombItem` will enable the Bandit to pick up a bomb item, i.e. changing the value of the appropriate world state key. If for example the agent lacks bomb item x but sees it, this action makes the agent pick up the bomb item, and thereby changing the Bandit's world state key `kHasBombItemX` from `false` to `true`.

  Preconditions: `(kSeesBombItemX, true) OR (kSeesBombItemY, true) OR (kSeesBombItemZ, true) OR (kSeesBombItemW, true)`

  Effects: `(kHasBombItemX, true) OR (kHasBombItemY, true) OR (kHasBombItemZ, true) OR (kHasBombItemW, true)`

- **Go to make bomb area**. The action `GoToMakeBombArea` will enable the Bandit to go to the make bomb area (see Figure 11) once all bomb items have been collected.

  Preconditions : `(kHasBombItemX, true) AND (kHasBombItemY, true) AND (kHasBombItemZ, true) AND (kHasBombItemW, true)`

  Effect: `(kIsAtMakeBombArea, true)`

- **Make bomb**. By using the action `MakeBomb` at the make bomb area, the Bandit will turn the bomb items into a bomb.

  Precondition: `(kIsAtMakeBombArea, true)`

  Effect: `(kHasBomb, true)`

- **Go to plant bomb area**. Much like the action `GoToMakeBombArea`, this action will enable the Bandit to go to the plant bomb area (see Figure 11) once the Bandit possess a bomb.

  Precondition: `(kHasBomb, true)`

  Effect: `(kIsAtPlantBombArea, true)`

- **Plant bomb**. By using the action `PlantBomb` at the plant bomb area, the Bandit plants the bomb.

  Precondition: `(kIsAtPlantBombArea, true)`

  Effect: `(kTargetIsBlownUp, true)`

5.3 Identified world state keys

The value type of all world state keys will be Boolean since more advanced value types are deemed unnecessary for this project. For instance will the world state key `kSeesPaper` have the value `true` or `false`, instead of a handle to the paper being seen.

To string together the bomb items with more accessible and intuitive real world counterparts, the naming of the bomb items will hereafter be Paper, Fertilizer, Sulphur and Bombshell. As the author is no bomb expert, it is at the reader's discretion to look upon the bomb items as he or she may see best fit. The following world state keys have been identified and are to be used in the project:

```
kSeesPaper
kSeesFertilizer
kSeesSulphur
kSeesBombshell
kHasPaper
kHasFertilizer
kHasSulphur
kHasBombshell
kIsAtMakeBombArea
kHasBomb
kIsAtPlantBombArea
kTargetIsBlownUp
kIsAtTargetNode
kSeesBandit
kSeesThreat
```

5.4 Further details of the architecture

Here the architecture's components are further explained and examples of their most interesting functions are given in pseudocode. The actions and goals used in this project will be implemented as separate classes and as Singletons, in order to have multiple entities use the same instance of a specific action or goal. To let each entity behave in its own way, the entity will be sent as an argument to the individual functions of the actions and goals. The preconditions and effects of an action will be implemented as a `struct` having a world state key (an enum) and a Boolean value.

- **The `Update` function of the working memory**. This function will update each memory fact and if it is too old it will be removed, i.e. forgotten.

```
for each memory fact
   add time to memory fact
   if fact is too old
      remove memory fact
```

- **The `Update` function of the seeing sense**. This function will update its owner's world states whether new entities are seen or not.

```
set copy of owner's relevant world state keys to false
for each seen entity
   identify entity type and set corresponding world state key
   add a working memory fact of the entity to the working memory
update owner's world states with local copy
```

- **The `Update` function of the agents**. This function will check whether the agent has reached its target node and whether the current plan is valid. It will also update the working memory and seeing sense.

```
update seeing sense
update working memory
if at target node
   (kIsAtTargetNode, true)
else
   (kIsAtTargetNode, false)
if current plan is valid
   carry out current plan
else
   ask the GOAP planner to formulate and send a new plan
```

- **The `FindMostRelevantGoal` function of the goal planner**. This function will ask each of the goal planner's goals its current relevance value.

```
for each goal loaded into the goal planner
   calculate the goal's relevance value
return the goal with highest relevance value
```

- **The `FormulatePlan` function of the GOAP planner**. This function will try to find a set of actions which enable its owner to reach the desired goal world state, as depicted in Figure 13.

```
get the current goal's preconditions
while there still are unmet preconditions
   for each action loaded into the GOAP planner
      for each effect of the current action
         for each unmet precondition
            if current effect matches current precondition
               has found a viable action
               remove current precondition from unmet list
      if found viable action
         add action to plan
         add action's preconditions to unmet preconditions
   if no viable action has been found
      return no plan could be formulated
```

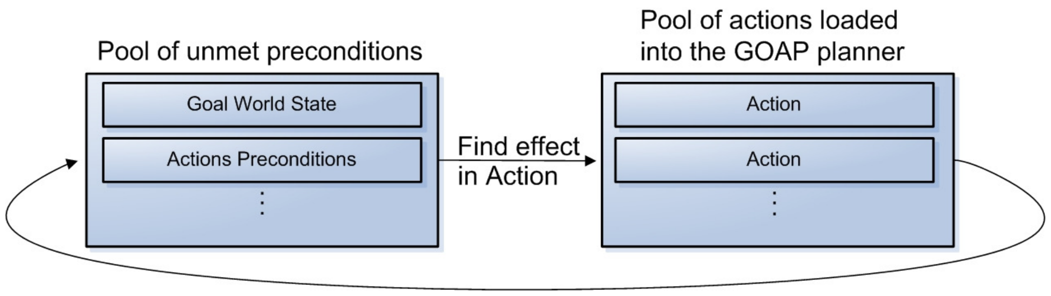

Pool of unmet preconditions
Goal World State
Actions Preconditions
Find effect in Action
Pool of actions loaded into the GOAP planner
Action
Action

Figure 13. The search of appropriate actions will continue until no more unmet preconditions exist, or no viable action could be found.

- **The `CarryOutPlan` function of a plan**. This function will handle the actions of a plan and can determine whether a plan is invalid or completed.

```
if current action is invalid
   return unable to carry out action
else if current action is completed
   if all the plan's actions have been executed
     return done
   else
     deactivate current action
     remove current action from plan
     activate next action
```

- **The `FindWorldStateThreats` function of the threat analyzer**. This function will create a matrix containing all world states deemed dangerous regarding the anti goal world state given, as depicted in Figure 12. As the matrix is made up of two vectors, no empty spaces are present.

```
get a list of world states deemed dangerous
while there still are dangerous world states
   for each action loaded into the threat analyzer
      for each effect of the current action
         while no dangerous action has been found
```

```
                for each unmet precondition
                   if current effect matches current precondition
                     has found dangerous action
                     add current action to danger level list
       for each dangerous action at this level
          add action’s preconditions to list of dangerous world states
```

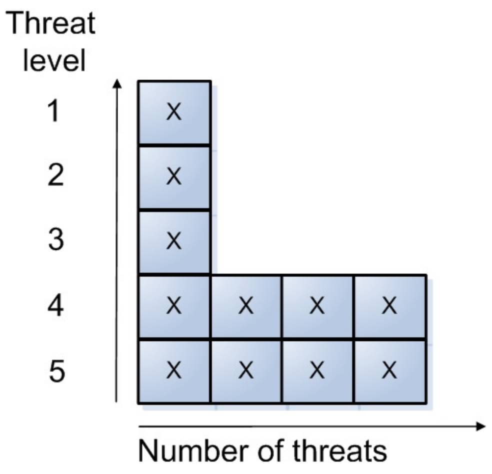


Figure 14. The matrix containing all world states deemed dangerous in this project, given the anti goal world state `(kTargetIsBlownUp, true)`.

- **The `Activate` function of the `AssessBandit` action**. This function will check whether the assessed Bandit has any of the world states deemed dangerous by the threat analyzer. In case dangerous world states are found, also the degree of that threat level’s completion will be given. In Figure 15 and Figure 16 two different examples are shown.

```
for each threat level of known threats, i.e. list of world states
   for each threat at current threat level, i.e. a world state
      if threats value equals world state key value of bandit
        tag bandit with current threat level and completion %
```

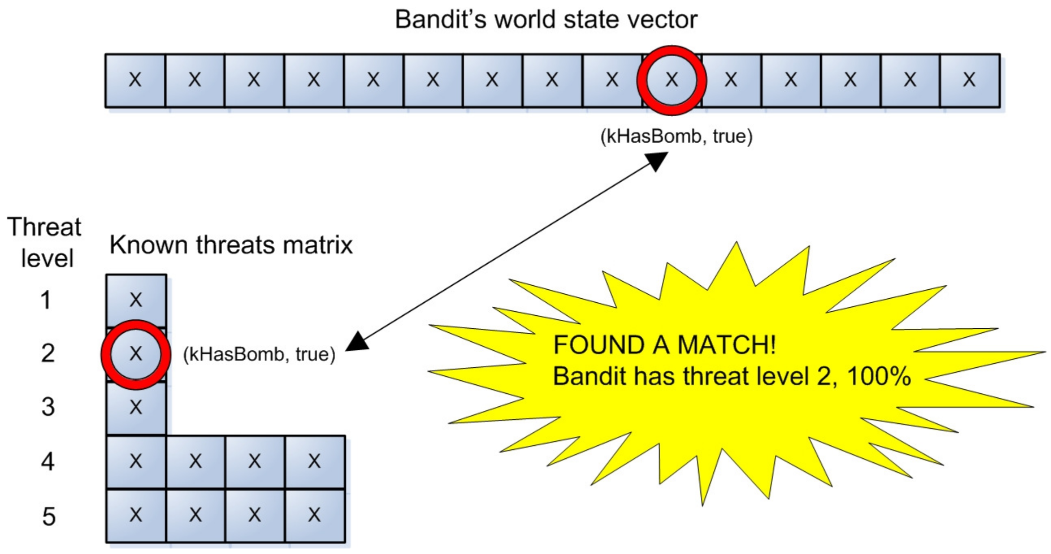


Figure 15. One of the known dangerous world states was found in the Bandit.

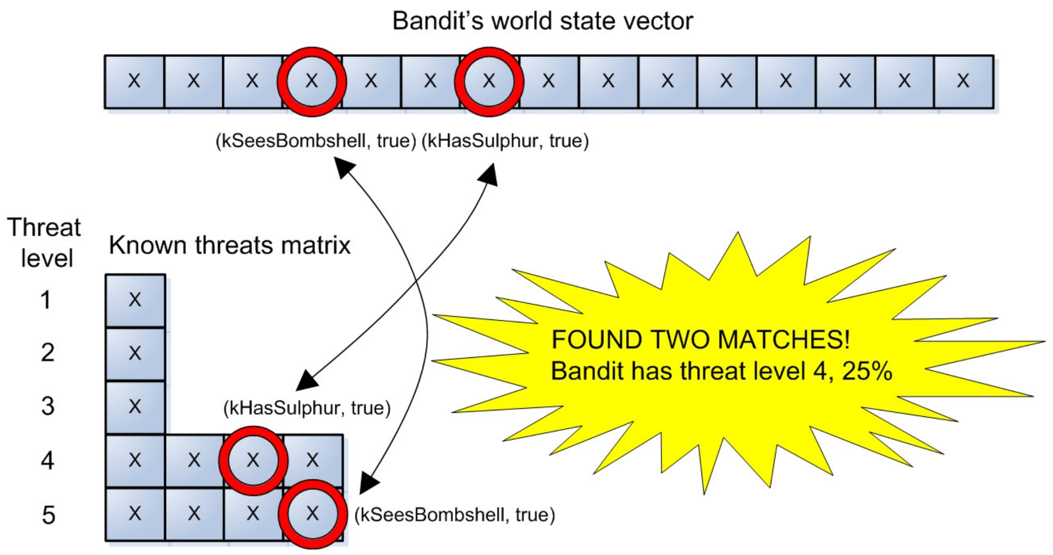

Bandit's world state vector
X X X X X X X X X X X X X X X
(kSeesBombshell, true) (kHasSulphur, true)
Threat level
Known threats matrix
1
2
3
4
5
(kHasSulphur, true)
(kSeesBombshell, true)
FOUND TWO MATCHES!
Bandit has threat level 4, 25%

Figure 16. Two of the known dangerous world states were found in the Bandit.

5.5 Scenario rules and conditions

The world size will be set to 800x600 pixels and the agents are to be given a movement speed of 100 pixels per second. This will enable the agents to go around their business fast enough while allowing an observer to follow the agents' behaviors. The agents seeing senses will be given a 90 degree field of view with a see range of 100 pixels. The field of view will thus somewhat map to a human counterpart, while the short see range keeps the world and scenario more manageable. To clearly see the agents and the bomb items, the entities are given a radius size of 10 or 15 pixels respectively.

In order to give the LECOF a chance to assess the Bandit while it is making or planting the bomb, the make and plant bomb actions will be given a duration of two seconds. Since the LECOF only assesses and never intervenes, the Bandit agent will at some point reach its goal. When this happens the scenario will be restarted and a new *round* will begin. The positions where the Bandit will be able to make and plant the bomb will be referred to as *The Shack* and *The Target* (see Figure 11).

5.6 Class diagram

Here two class diagrams are shown. The first diagram in Figure 17 is meant as an overview showing how the projects different parts connect. More detailed information of the actual implementation is then shown in Figure 18.

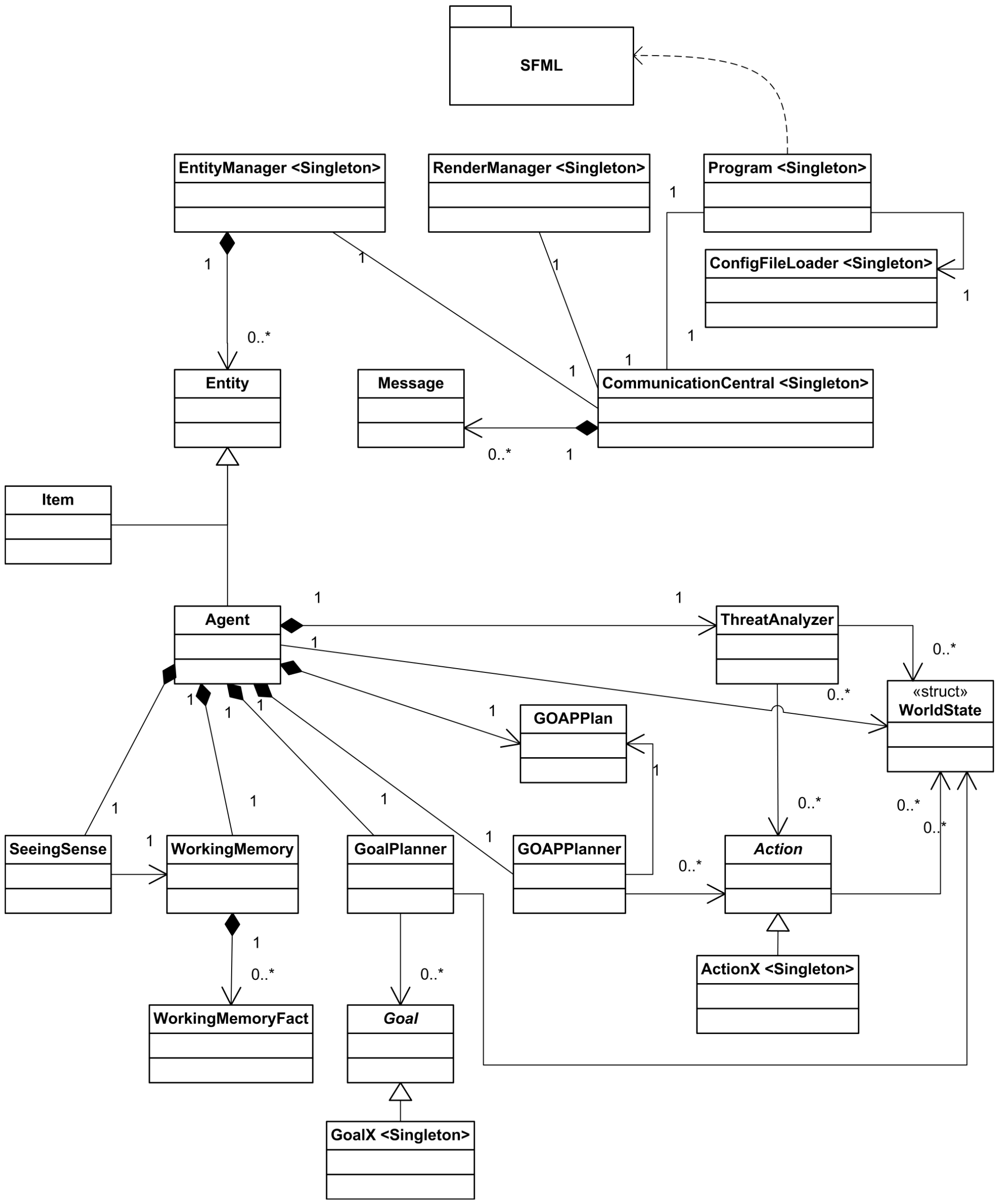


Figure 17. Overview class diagram.

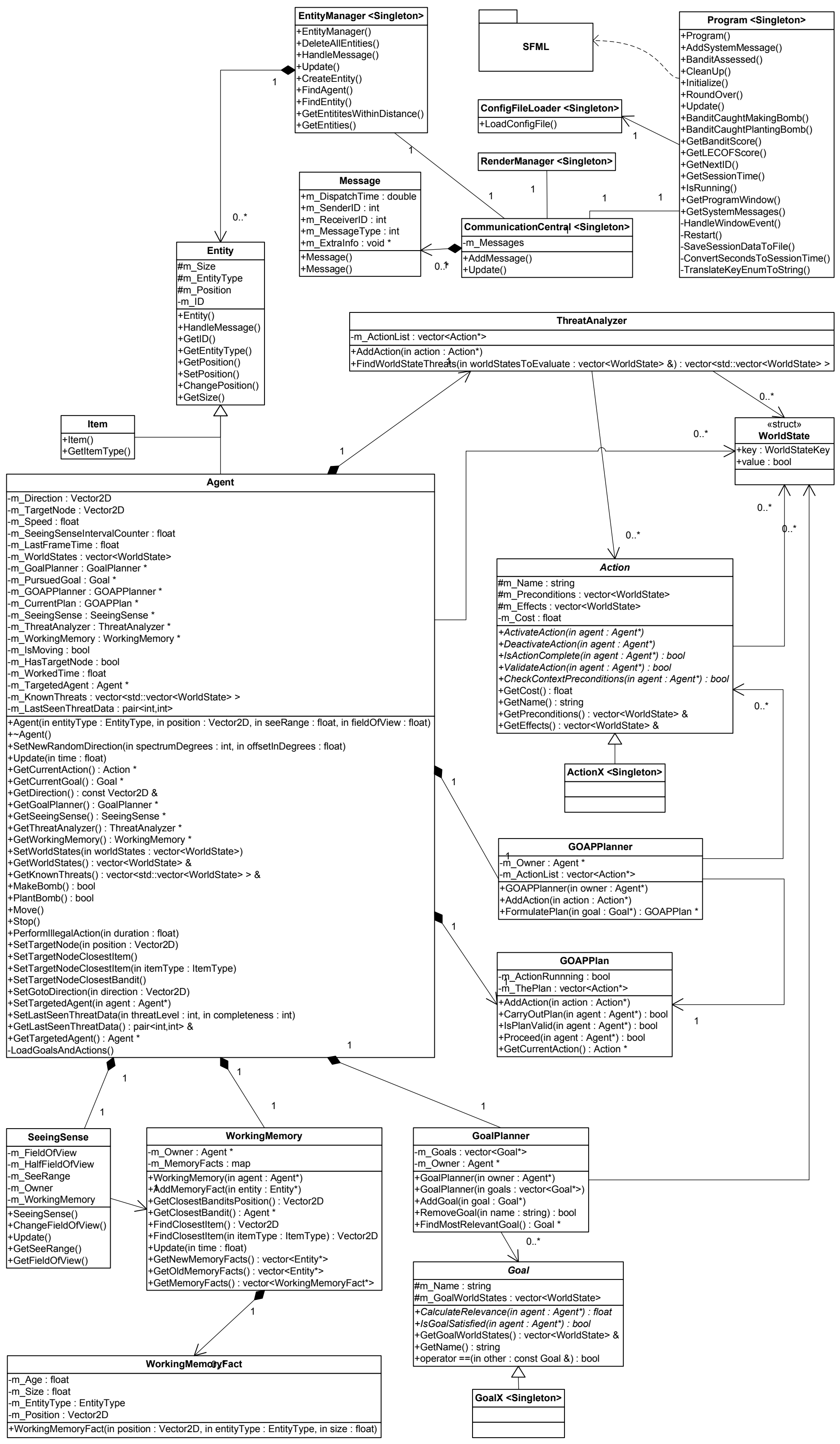

EntityManager <Singleton>
+EntityManager()
+DeleteAllEntities()
+HandleMessage()
+Update()
+CreateEntity()
+FindAgent()
+FindEntity()
+GetEntititesWithinDistance()
+GetEntities()
SFML
Program <Singleton>
+Program()
+AddSystemMessage()
+BanditAssessed()
+CleanUp()
+Initialize()
+RoundOver()
+Update()
+BanditCaughtMakingBomb()
+BanditCaughtPlantingBomb()
+GetBanditScore()
+GetLECOFScore()
+GetNextID()
+GetSessionTime()
+IsRunning()
+GetProgramWindow()
+GetSystemMessages()
-HandleWindowEvent()
-Restart()
-SaveSessionDataToFile()
-ConvertSecondsToSessionTime()
-TranslateKeyEnumToString()
ConfigFileLoader <Singleton>
+LoadConfigFile()
RenderManager <Singleton>
Message
+m_DispatchTime : double
+m_SenderID : int
+m_ReceiverID : int
+m_MessageType : int
+m_ExtraInfo : void *
+Message()
+Message()
CommunicationCentral <Singleton>
-m_Messages
+AddMessage()
+Update()
Entity
#m_Size
#m_EntityType
#m_Position
-m_ID
+Entity()
+HandleMessage()
+GetID()
+GetEntityType()
+GetPosition()
+SetPosition()
+ChangePosition()
+GetSize()
Item
+Item()
+GetItemType()
ThreatAnalyzer
-m_ActionList : vector<Action*>
+AddAction(in action : Action*)
+FindWorldStateThreats(in worldStatesToEvaluate : vector<WorldState> &) : vector<std::vector<WorldState> >
«struct»
WorldState
+key : WorldStateKey
+value : bool
Agent
-m_Direction : Vector2D
-m_TargetNode : Vector2D
-m_Speed : float
-m_SeeingSenseIntervalCounter : float
-m_LastFrameTime : float
-m_WorldStates : vector<WorldState>
-m_GoalPlanner : GoalPlanner *
-m_PursuedGoal : Goal *
-m_GOAPPlanner : GOAPPlanner *
-m_CurrentPlan : GOAPPlan *
-m_SeeingSense : SeeingSense *
-m_ThreatAnalyzer : ThreatAnalyzer *
-m_WorkingMemory : WorkingMemory *
-m_IsMoving : bool
-m_HasTargetNode : bool
-m_WorkedTime : float
-m_TargetedAgent : Agent *
-m_KnownThreats : vector<std::vector<WorldState> >
-m_LastSeenThreatData : pair<int,int>
+Agent(in entityType : EntityType, in position : Vector2D, in seeRange : float, in fieldOfView : float)
+~Agent()
+SetNewRandomDirection(in spectrumDegrees : int, in offsetInDegrees : float)
+Update(in time : float)
+GetCurrentAction() : Action *
+GetCurrentGoal() : Goal *
+GetDirection() : const Vector2D &
+GetGoalPlanner() : GoalPlanner *
+GetSeeingSense() : SeeingSense *
+GetThreatAnalyzer() : ThreatAnalyzer *
+GetWorkingMemory() : WorkingMemory *
+SetWorldStates(in worldStates : vector<WorldState>)
+GetWorldStates() : vector<WorldState> &
+GetKnownThreats() : vector<std::vector<WorldState> > &
+MakeBomb() : bool
+PlantBomb() : bool
+Move()
+Stop()
+PerformIllegalAction(in duration : float)
+SetTargetNode(in position : Vector2D)
+SetTargetNodeClosestItem()
+SetTargetNodeClosestItem(in itemType : ItemType)
+SetTargetNodeClosestBandit()
+SetGotoDirection(in direction : Vector2D)
+SetTargetedAgent(in agent : Agent*)
+SetLastSeenThreatData(in threatLevel : int, in completeness : int)
+GetLastSeenThreatData() : pair<int,int> &
+GetTargetedAgent() : Agent *
-LoadGoalsAndActions()
Action
#m_Name : string
#m_Preconditions : vector<WorldState>
#m_Effects : vector<WorldState>
-m_Cost : float
+ActivateAction(in agent : Agent*)
+DeactivateAction(in agent : Agent*)
+IsActionComplete(in agent : Agent*) : bool
+ValidateAction(in agent : Agent*) : bool
+CheckContextPreconditions(in agent : Agent*) : bool
+GetCost() : float
+GetName() : string
+GetPreconditions() : vector<WorldState> &
+GetEffects() : vector<WorldState> &
ActionX <Singleton>
GOAPPlanner
-m_Owner : Agent *
-m_ActionList : vector<Action*>
+GOAPPlanner(in owner : Agent*)
+AddAction(in action : Action*)
+FormulatePlan(in goal : Goal*) : GOAPPlan *
GOAPPlan
-m_ActionRunnning : bool
-m_ThePlan : vector<Action*>
+AddAction(in action : Action*)
+CarryOutPlan(in agent : Agent*) : bool
+IsPlanValid(in agent : Agent*) : bool
+Proceed(in agent : Agent*) : bool
+GetCurrentAction() : Action *
SeeingSense
-m_FieldOfView
-m_HalfFieldOfView
-m_SeeRange
-m_Owner
-m_WorkingMemory
+SeeingSense()
+ChangeFieldOfView()
+Update()
+GetSeeRange()
+GetFieldOfView()
WorkingMemory
-m_Owner : Agent *
-m_MemoryFacts : map
+WorkingMemory(in agent : Agent*)
+AddMemoryFact(in entity : Entity*)
+GetClosestBanditsPosition() : Vector2D
+GetClosestBandit() : Agent *
+FindClosestItem() : Vector2D
+FindClosestItem(in itemType : ItemType) : Vector2D
+Update(in time : float)
+GetNewMemoryFacts() : vector<Entity*>
+GetOldMemoryFacts() : vector<Entity*>
+GetMemoryFacts() : vector<WorkingMemoryFact*>
GoalPlanner
-m_Goals : vector<Goal*>
-m_Owner : Agent *
+GoalPlanner(in owner : Agent*)
+GoalPlanner(in goals : vector<Goal*>)
+AddGoal(in goal : Goal*)
+RemoveGoal(in name : string) : bool
+FindMostRelevantGoal() : Goal *
Goal
#m_Name : string
#m_GoalWorldStates : vector<WorldState>
+CalculateRelevance(in agent : Agent*) : float
+IsGoalSatisfied(in agent : Agent*) : bool
+GetGoalWorldStates() : vector<WorldState> &
+GetName() : string
+operator ==(in other : const Goal &) : bool
GoalX <Singleton>
WorkingMemoryFact
-m_Age : float
-m_Size : float
-m_EntityType : EntityType
-m_Position : Vector2D
+WorkingMemoryFact(in position : Vector2D, in entityType : EntityType, in size : float)
1
0..*
1
0..*

Figure 18. Detailed class diagram.

6 Results

This chapter presents the results and an analysis of what have been observed during the implementation and execution of the architecture, according to objective 1 and 2 presented in chapter 3.

As in most simulations the result is highly coupled with what starting values and abilities the performing entities are given. In this project parameters such as world size, item placement, agent speed, field of view, see range, entity size and other ability limitations such as always looking in the direction of movement, have a critical impact on the overall result. By tweaking these parameters to a somewhat realistic or accessible scenario, the result can be evaluated in the context of the problem at hand. The point here is that the values that were used do share a resemblance of a real world counterpart, even though they have been tweaked to serve the outcome in a positive way. For example were the agents' field of view set to 90 degrees and their movement speed to 100 pixels per second. If the LECOF's field of view was to be impaired to zero, i.e. turning the LECOF blind, the Bandit would never be assessed and no threat assessments would occur, even though the underlying threat analysis technique remained the same. As it turned out, the scenario was simplified enough to be accessible, whilst remaining true to a somewhat similar real world setting.

6.1 Results from objective 1: Development of the necessary components

The implementation of the architecture was straightforward and no real set backs occurred during development. Some thought had to given to the amount of goal abstraction; even though it would be perfectly possible to create *goals* like `FindBombItem` and `PickUpBombItem`, the planning benefits would be lost as each goal would only replace the other as progress is made. In this project, the whole scenario could be covered with just one goal per agent, as for example the goal `PlantBomb` (i.e. `(kTargetIsBlownUp, true)`) did the job of linking all actions leading to the bomb being planted.

As the actions and goals implemented inheritance, it was easy to create new action or goal classes as development progressed. When a new action was to be implemented, it simply inherited the abstract base class and specified how it was supposed to be carried out. This way behavior is easily reused, as also pointed out by Borgers et al. (2008) in their 2APL implementation.

A somewhat tricky part of the development was to identify the world states and actions that enable an agent to do what you as a developer wants it to do; while at the same time doing it as optimal as possible. This of course depends on the scenario at hand, since the making of a bomb could have several preconditions and actions that have to be met prior to making the bomb. It thus lies in the nature of Goal-Oriented Action Planning (GOAP) (i.e. STRIPS) to be used in a concentrated area of interest. It would probably be impossible to describe the entire world in GOAP terms, since there would be an almost endless amount of world states. For example, a human face could be described with world states of each skin cell's color, position and age; or it could be described with just one world state key paired with an enum, stating whether the face has a smile or grin. It is all up to the purpose of the application and what is to be modelled, and therein lies the difficulty of determining the level of detail.

An interesting observation during development was the actions' close resemblance of that of states in a finite state machine (see section 2.2.1). Table 1 shows how the FSMs' functions directly map to the actions' functions. For example are the action functions `Activate` and `Deactivate` run once in order to prepare the action for its execution, or to tidy up after its execution; just as a state in a FSM implementation. Furthermore, the action functions `ValidateAction`, `IsActionComplete`, and `CheckContextPreconditions` somewhat corresponds to the FSMs `Execute` function, which is run with a suitable update frequency.

FSM function	**Action function**
`Enter`	`Activate`
`Execute`	`ValidateAction,IsActionComplete,CheckContextPreconditions`
`Exit`	`Deactivate`

Table 1. This table shows how regular FSM functions directly map to some of those of an abstract action.

The design of the threat analyzer heavily resembles that of the GOAP planner, even though the analyzer builds an entire matrix of dangerous world states (see Figure 14). This matrix construction could in a larger scenario with many actions (and consequently many world states) take a considerable amount of time, which may seem undesirable in an ever changing world. However, since the anti goals of the threat analyzer probably will be quite few and under the precursor that no new actions are considered at runtime, the matrices could be built in advance, i.e. not at runtime.

As said earlier the overall implementation was straightforward and the primary strength during implementation must be the modularity of the planners' build-up. Once the framework of a planner or threat analyzer is in place, its behavior is made up from goals and actions, which in turn have a high amount of modularity as world states make up the core of an individual goal or action. If, for example, the Bandit also should be able to *buy* a bomb, as well as to make it, a `BuyBomb` action could easily be added and loaded into the GOAP planner with the same effect as `MakeBomb`, i.e. `(kHasBomb, true)`. If the Bandit had to have money and be at a certain place to buy the bomb, precondition like `(kHasMoney, 1000)` and `(kIsAtBombMarket, true)` could be employed. Since the size of this project did not call for all GOAP components, a few of them could be omitted while still being able to utilize the core of the architecture, which per se could be seen as an architectural strength.

Even though the STRIPS part of GOAP offers a modular and reusable approach, it seems like the goal planner, responsible of finding the goal world state, could quickly become cumbersome and hard to overview depending on how the relevancy value is calculated. If the GOAP architecture is to be part of a complex system, the Hierarchical Task Network (HTN) model might be necessary in order to have non-primitive tasks take care of the order of actions, even though the goal mapping between STRIPS and HTN can be an intricate matter, as pointed out by Paolucci et al. (2000). A complex dynamic system also implicates fast planning where a non-anytime planner like GOAP might be less favored over a planner employing forward searches. Furthermore, the action linkage from preconditions and effects in GOAP could become hard to justify or keep accurate if numerous actions and world states are present; so some kind of tool that gives an overview of legal connections would probably be useful in such a large system.

6.2 Results from objective 2: Evaluation of gathered data and observed behavior

An analysis of an excerpt of the data logs from a three day session (1.883 rounds) is given in section 6.2.1, while section 6.2.2 gives a walk-through of an arbitrary round. The session is considered to be representative for the implementation, and corresponds to the characteristics of the scenario being modeled.

The LECOF agent using the threat analyzer based on GOAP is without doubt able to correctly assess the Bandit's threat level and its completion. It was impressive to witness how the threat analyzer was capable of noticing dangers in world states that had not been individually tagged as dangerous; much like the strengths of the GOAP planner. The threat analyzer does indeed seem like a powerful tool since only by deeming a single world state dangerous, the threat analyzer does not only find all[18] dangerous world states, but it also finds somewhat *how* dangerous the assessed entity is. As the algorithm employed for building the threat matrix uses brute force searches, it could be optimized in a number of ways. The same heuristics optimizations normally used by the GOAP planner, could probably be used for the threat analyzer[19].

Due to the placement of the bomb items and the given field of view and see range, it was impossible for the Bandit agent to exceed 25 % completion of the level 5 threat; that is, the Bandit agent was never capable of seeing more than one bomb item at a time. Because of the simplicity of this scenario, it is possible to directly map a certain threat level and completion to a certain world state. This would not be possible, nor would it be necessary, in a more complex scenario consisting of more actions and world states.

In a vast scenario where large amounts of actions and world states exist, the individual threats (world states) could be given a dynamic cost, much like how an action may be given a certain cost, which then is used by the planner to select the most promising actions. For instance, in a scenario where a forest fire `(kForestIsAflame, true)` is the anti goal of the threat analyzer, the world state `(kHasLighter, true)` could be given a higher cost, i.e. seen as a bigger threat, if the world state key `kSoilIsDry` has the value `true`, i.e. `(kSoilIsDry, true)`.

6.2.1 Analysis of data log excerpts

During the three day session 1.883 rounds were simulated. The data logs are to show how the Bandit assessments can serve as a foundation of strategic and tactical management, enabling the resources at hand, i.e. the Law Enforcing Coalition Unit(s), to be used more efficiently. Since the LECOF agent never intervened or did anything to stop the Bandit agent, the Bandit's goal world state was always reached since the Bandit was able to plant the bomb soon enough. Table 2 shows the sessions' assessment data, where each type of threat and its completion is shown with how many rounds it occurred in, and how many times it was seen on the whole.

[18] With respect to the actions loaded into the threat analyzer.

[19] Normally the pointers to the actions are kept in matrix (a vector in a vector) indexed by preconditions or effects. Each position thus keeps a vector of all actions having that precondition or effect.

THREAT LEVEL	**COMPLETION**	**OCCURANCE (# of rounds)**	**TOTAL TIMES FOUND (all rounds)**
NO THREAT	-	437 (23.2%)	1095
1	100 %	143 (7.6%)	144
2	100 %	221 (11.7%)	221
3	100 %	137 (7.3%)	137
4	25 %	808 (42.9%)	1941
4	50 %	1002 (53.2%)	3453
4	75 %	1295 (68.8%)	5441
4	100 %	352 (18.7%)	491
5	25 %	94 (5.0%)	272

Table 2. Data from a three day session (1.883 rounds).

Since some of the Bandit's world states are "occurring" only for a short while of time, the LECOF was only capable of catching certain threats, if lucky enough. An example of this is threat level 2 which in this scenario directly maps to the world state `(kHasBomb, true)`. The Bandit possesses this world state only for a short while, as it is travelling from The Shack to The Target. In addition to this, the behavior[20] of the LECOF is also responsible of not being able to find this threat more than once each round.

The data further shows that in about 23.2 % of the rounds, the LECOF assessed the Bandit before it had picked up any bomb items. Depending on how we want the scenario to turn out, the LECOF's tolerance could be set to a certain threat level and completion. This would allow the LECOF to follow the Bandit from, let's say, threats of level 4 with 75 % completion, but sound an alarm if threat level 2 with 100 % completion was spotted. If this doctrine was employed, the data logs show us that the LECOF would have followed the Bandit in about 68.8 % of the scenarios, and then at some point, arrest or in other ways stop the Bandit from reaching its goal. This of course assumes that the Bandit stays somewhat stupid, and that neither the Bandit nor the LECOF change their current behavior once the Bandit is spotted.

Overall the data represents what could be anticipated with the current scenario layout. For example can the huge presence dip between threat level `(4, 75%)` and `(4, 100%)` be explained by the change of the Bandit's behavior once all bomb items have been collected. For the LECOF to spot the threat `(4, 100%)`, it must be lucky enough to see the Bandit as the Bandit travels from the location of the last bomb item to The Shack. If the threat handling rules were changed, lower threats could be more frequent in the statistics, i.e. this implementation does only regard the highest threat reached (the lowest threat level value), which of course has an impact on the statistics. If this handling was changed, the threat `(5, 25%)` would have a higher presence since it would be regarded even though the bandit was carrying one or more bomb items, i.e. having threat `(4, 25%)` to `(4, 75%)`.

6.2.2 A walk-through of an arbitrary round

The figures (screenshots) from this walk-through are located in the Appendix.

[20] Once the LECOF has assessed the Bandit, the LECOF turns 180 degrees and continues its search.

As the simulation starts the two agents are facing away from each other and moving in the direction they are facing, the LECOF being to the left (Figure 19). The agents have been loaded with the appropriate goals and actions described in section 5.2. Once the agents have reached the walls of the world a new direction is randomly selected (Figure 20). As the agents travel in their new randomly selected direction, the LECOF agent stumbles across the Bandit agent and performs an assessment, indicating that the Bandit is of no threat (Figure 21 and Figure 22). The LECOF's action change from `FindBandit` to `AssessBandit` happened as the `FindBandit` action completed, which in turn caused the LECOF's seeing sense to change the world state `kSeesBandit` from `false` to `true`. Since the `AssessBandit` action completes immediately, all actions of the LECOF's plan have been executed, causing the goal planner to once again look for the most promising goal; which then is fed to the GOAP planner that returns a new plan to the LECOF agent. Because of the limited amount of goals and actions, the returned plan is identical to the one formulated last time, i.e. the plan consists of the actions `FindBandit` and `AssessBandit`.

Soon enough the Bandit's seeing sense notices the Paper item, which the Bandit does not possess, which leads to the completion of the current action `FindBombItem` and activation of the plan's next action `PickUpBombItem` (Figure 23). When the Bandit is in reach of the Paper item, it gets picked up and the `PickUpBombItem` action completes. This leads to an activation of the next action `GoToMakeBombArea`, but since that action has the precondition of the Bandit having all four bomb items, the action is invalidated and a new plan is formulated[21]. The first action `FindBombItem` of the new plan is then activated, and the Bandit once again starts its search for bomb items (Figure 24). A while later, chance has it that the LECOF and Bandit are moving towards each other, leading to another assessment of the Bandit. Since the Bandit now possess the Paper item, the LECOF agent determines that the Bandit is of threat level 4 with 25 % completion (Figure 25 and Figure 26).

At about 46 seconds into the simulation, the Bandit spots the Bombshell item and acts just like when the Paper item was spotted and picked up. As the LECOF agent is heading towards the Bandit, another assessment takes place. Since the Bandit now holds two items (Paper and Bombshell), the Bandit is tagged with threat level 4 with 50 % completion (Figure 27 through Figure 29).

After one minute and six seconds, the Bandit happens to spot the Fertilizer item and performs the same actions and re-planning as before, ending up searching for the last bomb item (Figure 30 and Figure 31). On its journey to the last bomb item, the Bandit is once again intercepted by the LECOF, who this time sees the Bandit carrying three bomb items, and determines that the Bandit is of threat level 4 with 75 % completion (Figure 32 and Figure 33). A few seconds later, the Bandit hits the wall and ends up in a direction towards the Sulphur item. Once the Bandit spots the Sulphur, it gets picked up and the `PickUpBombItem` action completes. Since the Bandit now possess all four bomb items, the following action `GoToMakeBombArea` does not get invalidated; whereby the Bandit sets off towards The Shack where the bomb is to be made (Figure 34 through Figure 36).

[21] Just like the chain of events that occurred when the LECOF agent ran out of planned actions; the Bandit's goal planner is first called which returns a goal world state, which then is fed to the GOAP planner which in turn comes up with a new plan.

When the Bandit arrives at The Shack, the bomb making starts to take place as the action `GoToMakeBombArea` completes and `MakeBomb` activates. But before the Bandit completes the bomb[22], it is once again assessed by the LECOF, who this time determines the Bandit to be of threat level 3 with 100 % completion. A short while later, the Bandit finishes the bomb and heads toward The Target where the bomb is to be planted. This means that after the `MakeBomb` action was completed, the `GoToPlantBombArea` action was activated. Seconds later the Bandit arrive and plants the bomb by completing the action `GoToPlantBombArea` and activating the action `PlantBomb`; which ends the round since the Bandit's goal world state `(kTargetIsBlownUp, true)` has been reached (Figure 37 through Figure 41).

Because this walk-through has described an actual round taking place, some threat levels were not detected by the LECOF agent. Figure 42 through Figure 44 depict the assessment types that didn't occur in this walk-through. The figures show how the Bandit is assessed when spotting a bomb item[23] (threat level 5 with 25% completion), when having a bomb (threat level 2 with 100% completion), and planting a bomb (threat level 1 with 100% completion).

[22] The making and planting of the bomb were set to two seconds.

[23] The Bandit was not carrying any bomb items when the LECOF assessed the Bandit.

7 Conclusion

The simplified Goal-Oriented Action Planning (GOAP) architecture was a straightforward implementation and the incorporated actions had clear relationships with states that reside in a typical finite state machine. The threat analyzer performed accurately as planned and was able to determine the threat level and completion of the assessed entity, the drawback being its matrix construction which in a complex system might take considerable time; which in turn can be avoided by building the threat matrix offline. Its usefulness depends heavily on the detail of the scenario, i.e. what world states and transitions (actions) are being modeled, and on what basis eventual action costs are calculated. It was however impressive to witness how the threat analyzer was capable of noticing dangers in world states that had not been individually tagged as dangerous.

Some examples have been given of how the novel human-to-human, human-to-machine, and machine-to-machine Coalition Battle Management Language could be used in the context of this project. Furthermore, by introducing Figure 1 and Figure 4 the community has been given new views of conceptual schemas describing a styled agent that uses a reasoning system, and how the blackboard concept fits with the OODA-loop.

The area of planning is without doubt a complex matter with almost endless application areas and techniques. This thesis has contributed by coherently presenting both profound and novel key areas of planning while at the same time building and examining an actual implementation. The project has produced a small sandbox capable of handling agents using a simplified GOAP architecture and threat analysis based on the same technique. Even in this basic implementation it is easy to make the threat assessing agent fire a certain behavior once an adequate threat has been found, or to evaluate where available resources are best put into action.

7.1 Research discussion

As described in section 6.1 the world states used in an arbitrary simulation heavily depends on what is to be modeled. The decision on what world states should be modeled could become a substantial task when a large system is to be employed. Much like the information refinement when a decision is to be made from a massive information set, as described by Lagervik & Gustavsson (2006), the world states relevant to a simulation could perhaps be extracted by a similar data mining approach. If we had sensors capable of labeling relevant world states, both practical and subliminal, much of today's systems could more or less directly be taken into our world without changing their internal functions. It is an intriguing thought to picture how an agent from a first person shooter computer game could be realized into our world, if only it was equipped with sensors able to label the world states as the planning system demands; in addition to the need of an adequate motor system.

It is not hard to imagine a connection between GOAP and C-BML where the world states, goals, actions, and effects are represented and handled through BML's Command and Control Lexical Grammar (C2LG). As the Bandit agent's overall goal in this project was to plant a bomb, the WHO-WHEN-WHERE-WHY-WHAT representation could be employed to formulate an order leading to the anticipated end-state of a planted bomb. In this case a Bandit order "*plant bomb at target area*" would have "*plant bomb*" as WHAT, and "*at target area*" as WHERE. Of course could also the LECOF agent be commanded in a similar way given an order "*look for* *threats everywhere*". This technology or technique application, in conjunction with voice recognition, might become the very foundation of internal systems used by our robot companions of tomorrow; becoming a natural everyday tool, able to perform various tasks with social skills, enabling it to educate, entertain, exercise or otherwise serve its commander.

As one in the long run never can be completely sure about the effects of a formulated plan or executed action, an effect seems like more than a mere world state change, as it is able to start a chain reaction of more or less predicted events. It should also be noted that the observer's style determines what the effect really is and what effect derivatives subsequently emerge. If the effect stays the same but the observer changes his or her conception, it all boils down to if there exists an absolute truth or not, and whether that matters or not. The interesting conclusion here is that it is hard, if not impossible, to *measure* effect in an *unbiased* way since there exists none or almost infinite interpretations of what a certain world state implicates, depending on the *style* of the observer.

In a real world scenario years from now, the author would like to see how his commanded robot companion *Sprocket* is able to formulate and execute a plan, as he is given a verbal order of fetching a glass of water. As the robot co-worker sets off on his mission for a better world, the commander's verbal order has been interpreted through Sprocket's C2LG unit and his sensors register and label his surroundings into appropriate world states while his steering behavior keeps him away from both static and dynamic obstacles. Upon reaching the water faucet Sprocket finds the water line cut off, and after a quick re-plan he figures that it is better to return to his commander than go out and personally rearrange the garden and dig for water. As Sprocket returns he explains the situation through C2LG and is able to give a number of water alternatives, of which his commander settles for the *buy-water-at-nearby-store* alternative. Minutes later Sprocket returns with a mixture of hydrogen and oxygen, whereby his mission completes as his commander's thirst is quenched – all thanks to Sprocket's situation awareness, knowing and reasoning of world states and effects of his actions.

7.2 Future work

With limited resources a continued development of this project would try out a more complex world, adding more actions, world states and consequently more goals and anti-goals into the GOAP planner and threat analyzer. This would further test the analyzer's capabilities and judge its usability. With more resources at hand further development of squad behavior in conjunction with the threat analyzer and learning of new world states and actions would be evaluated, and maybe adding genetic algorithms[24] to the costs of actions, goals and threats would yield valid and interesting behaviors, i.e. styles.

If vast amounts of resources were available it could be possible to achieve something really spectacular if novel techniques like C-BML were fused with more or less established techniques built on STRIPS and HTN. Equipped with the pinnacle of subsystems in both hardware and software, a futuristic all-round robot companion with flawless embodiment, senses, and steering behaviors would become an epic achievement as it truly would constitute all areas of computer science and robotics.

[24] *Genetic algorithms* are described in detail by Buckland (2002).

Acknowledgements

This thesis was written at Saab Microwave Systems (SMW) in Skövde, Sweden. The author wants to thank Per M. Gustavsson and Eva Nero at SMW, and Mikael Thieme at the University of Skövde for their valuable guidance and support.

References

Ackoff, R. L. 1999. *Ackoff's Best – His Classic Writings on Management*. John Wiley & Sons, Hoboken, New Jersey.

Alberts, D. S. 2007. Agility, Focus, and Convergence: The Future of Command and Control. *The International C2 Journal*, 1(1), pp. 1-30. Available from: http://www.dodccrp.org/files/IC2J_v1n1_01_Alberts.pdf [Accessed 28 April 2008]

Aykroyd, P., Harper, K. A., Middleton, V. and Hennon, C. G. 2002. Cognitive Modeling of Individual Combatant and Small Unit Decision-Making within the Integrated Unit Simulation System. *In Proceedings of the 11th Computer-Generated Forces and Behavior Represe*, May 7-9, Orlando, FL, USA, ref 11TH-CGF-073.

Baxter, J. and Hepplewhite, R. 2000. A Hierarchical Distributed Planning Framework for Simulated Battlefield Entities. *In Proccedings of the 19th Workshop of the UK Planning and Scheduling Special Interest Group*, Milton Keynes, UK, December 13-14. Available from: http://mcs.open.ac.uk/plansig2000/Papers/P7.pdf [Accessed 4 April 2008]

Bedney, G. and Meister, D. 1999. Theory of Activity and Situation Awareness. *Int. J. Cognitive Ergonomics*, 3(1), pp. 63-72.

Blais, C., Galvin, K. and Hieb, M. 2005. Coalition Battle Management Language (C-BML) Study Group Report. *In Proceedings of the 2005 Fall Simulation Interoperability Workshop*, Orlando, Florida, USA, September, ref 05F-SIW-041 Available from: http://www.movesinstitute.org/~blais/Documents/05F-SIW-041.pdf [Accessed 4 February 2008]

Borgers, E., Spaans, M., Voogd, J., Hieb, M. R. and Bonse, R. 2008. Using a Command and Control Language to Simulate Operations in a Multi-Agent Environment. *In Proceedings of the 13th ICCRTS: C2 for Complex Endeavors*, Bellevue, WA, USA, June 17-19. [CD-ROM]

Breton, R. and Rousseau, R. 2005. The C-OODA: A Cognitive Version of the OODA Loop To Represent C^2 Activities [online]. *In Proceedings of the 10th International Command and Control Research and Technology Symposium*, McLean, USA, June 13-16. [CD-ROM] Available from:
http://www.dodccrp.org/events/10th_ICCRTS/CD/presentations/280.pdf
[Accessed 4 February 2008]

Buckland, M. 2002. *AI Techniques for Game Programming*. Premier Publishing Ltd, Glasgow, UK.

Buckland, M. 2005. *Programming Game AI by Example*. Wordware Publishing Inc, Plano, Texas, USA.

Buschmann, F. 1996. *Pattern-Oriented Software Architecture*. Wiley & Sons Ltd, West Sussex, UK.

Burke, R., Isla, D., Downie, M., Ivanov, Y. and Blumberg, B. 2001. Creature Smarts: The Art and Architecture of a Virtual Brain. *In Proceedings of Game Developers Conference*, San Jose, California, USA, March 20-24, pp. 147-166. Available from:
http://www.media.mit.edu/characters/additional%20resources/gdc01.pdf
[Accessed 21 February 2008]

Doris, K. and Silvia, D. 2007. Improved Missile Route Planning and Targeting using Game-Based Computational Intelligence. *In Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications*, Honolulu, Hawaii, April 1-5, pp. 63-68.

Dybsand, E. 2004. AI Game Programming Wisdom 2. In: Steve Rabin, editor. *Goal-Directed Behavior Using Composite Tasks*. Charles River Media, Massachusetts, chapter 3.6, pp. 237-245.

Endsley M. R. 1995. Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors*, 37(1), pp. 32-64.

Erol, K., Hendler, J. and Nau, D. S. 1994. HTN Planning: Complexity and Expressivity. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Seattle, Washington, USA, July 31-August 4, pp. 1123-1128.

Farrell, P. S. E. 2004. Measuring Common Intent during Effects Based Planning [online]. *In Proceedings of 2004 Command and Control Research and Technology Symposium: The Power of Information Age Concepts and Technologies*, Defence Research and Development Canada – Toronto, San Diego, CA, USA, June 15-17. [CD-ROM] Available from: http://www.dodccrp.org/files/2004_CCRTS.zip [Accessed 29 April 2008]

F.E.A.R. – First Encounter Assault Recon 2005. Monolith Productions/Sierra Entertainment Inc, Kirkland, Washington, USA. [Computer program]

Fikes, R. E. and Nilsson N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4), pp. 189-208.

Gustavsson, P. M., Nero, E., Wemmergård, J., Turnitsa, C., Korhonen, M., Evertsson, D., and Garcia, J. 2008. Tactical Situation Object – Enabling joint Crisis Management Training. *In Proceedings of the 2008 Spring Simulation Interoperability Workshop*, IEEE CS Press, Providence, Rhode Island, USA, April 14-17, ref 08S-SIW-018.

Gustavsson, P. M., Hieb, M., Eriksson, P., Moore, P. and Niklasson, L. 2008. Machine Interpretable Representation of Commander's Intent. *In Proceedings of the 13th ICCRTS: C2 for Complex Endeavors*, Bellevue, WA, USA, June 17-19, ref I-188. [CD-ROM]

Hieb, M. R. and Schade, U. 2008. A Linguistic Basis For Multi-Agency Coordination. *In Proceedings of the 13th ICCRTS: C2 for Complex Endeavors*, Bellevue, WA, USA, June 17-19, ref I-152. [CD-ROM]

Hieb, M. R. and Schade, U. 2006. Formalizing Battle Management Language: A Grammar for Specifying Orders. Presented at *Spring Simulation Interoperability Workshop*. Simulation Interoperability Standards Organization, Huntsville, AL, USA, April 2-7, ref 06S-SIW-068.

Hoang, H., Lee-Urban, S. and Muñoz-Avila, H. 2005. Hierarchical Plan Representations for Encoding Strategic Game AI. *In Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*, AAAI Press, June 1-5, Marina del Rey, CA, USA, pp. 63-68.

Holland, T. O. 2008. Towards a Lexicon for the Modeling and Simulation of Emergent Behavior Systems. *In Proceedings of 2008 Spring SIW/BRIMS Conference*, April 14-18, Providence, RI, USA, ref 08S-SIW-058.

International Game Developers Association, Working Group on Goal-Oriented Action Planning 2003. *The 2003 AIISC Report*. Available from: http://www.igda.org/ai/report-2003/aiisc_goap_report_2003.html
[Accessed 14 February 2008]

International Game Developers Association, Working Group on Goal-Oriented Action Planning 2004. *The 2004 AIISC Report*. Available from: http://www.igda.org/ai/report-2004/goap.html [Accessed 14 February 2008]

International Game Developers Association, Working Group on Goal-Oriented Action Planning 2005. *The 2005 AIISC Report*. Available from: http://www.igda.org/ai/report-2005/goap.html [Accessed 11 February 2008]

Kleiner, M. S., Carey, S. A. and Beach, J. 1998. Communication Mission-Type Orders To Virtual Commanders. In: D. J. Medeiros, E. F. Watson, J. S. Carson, M. S. Manivannan, editors. *In Proceedings of the 1998 Winter Simulation Conference*, Washington DC, USA, December 13-16, pp. 881-886.

Lagervik, C. and Gustavsson, P. M. 2006. A System Theoretical Approach to Situation Awareness and Architecture. *In Proceedings of the 11th International Command and Control Research and Technical Symposium (ICCRTS)*, Cambridge, UK, September 26-28. [CD-ROM]

Long, E. 2007. Enhanced NPC Behaviour using Goal Oriented Action Planning. M.Sc. thesis. School of Computing and Advanced Technologies, Division of Software Engineering, University of Abertay Dundee, Scotland, UK. Available from: www.edmundlong.com/Projects/Masters_EnhancedBehaviourGOAP_EddieLong.pdf
[Accessed 19 February 2008]

Muñoz-Avila, H. and Hoang, H. 2006. AI Game Programming Wisdom 3. In: Steve Rabin, editor. *Coordinating Teams of Bots with Hierarchical Task Network Planning*. Charles River Media, Massachusetts, chapter 5.4, pp. 417-427.

No One Lives Forever 2 : A Spy in H.A.R.M.'s *Way* 2002. Monolith Productions/Sierra Entertainment Inc, Kirkland, Washington, USA. [Computer program]

O'Brien, J 2002. AI Game Programming Wisdom. In: Steve Rabin, editor. *A Flexible Goal-Based Planning Architecture*. Charles River Media, Massachusetts, chapter 7.5, pp. 375-383.

Ontañón, S., Mishra, K., Sugandh, N. and Ram, A. 2007. Case-Based Planning and Execution for Real-Time Strategy Games. *In Proceedings of the International Conference on Case-based Reasoning (ICCBR 2007)*, Belfast, Northern Ireland, UK, August 13-16, pp. 164-178. Available from: http://www.cc.gatech.edu/faculty/ashwin/papers/er-07-11.pdf
[Accessed 11 February 2008]

Orkin, J. 2004a. Symbolic Representation of Game World State: Toward Real-Time Planning in Games. *In Proceedings of AAAI Workshop - Challenges in Game Artificial Intelligence 19th National Conference on Artificial Intelligence*, AAAI Press, San Jose, CA, USA, July 25-26, pp. 26-30.

Orkin, J. 2004b. AI Game Programming Wisdom 2. In: Steve Rabin, editor. *Simple Techniques for Coordinated Behavior*. Charles River Media, Massachusetts, chapter 3.2, pp. 199-206.

Orkin, J. 2004c. AI Game Programming Wisdom 2. In: Steve Rabin, editor. *Applying Goal-Oriented Action Planning to Games*. Charles River Media, Massachusetts, chapter 3.4, pp. 217-227.

Orkin, J. 2005. Agent Architecture Considerations for Real-Time Planning in Games. *In Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*, Marina del Rey, California, USA, June 1-3, pp. 105-110. Available from: http://web.media.mit.edu/~jorkin/aiide05OrkinJ.pdf [Accessed 21 February 2008]

Orkin, J. 2006. Three States and a Plan: The A.I. of F.E.A.R. *In Proceedings of the Game Developer's Conference 2006*, San Jose, CA, USA, March 20-24. Available from: www.cs.ualberta.ca/~bulitko/F06/papers/gdc2006_orkin_jeff_fear.pdf [Accessed 31 January 2008]

Paolucci, M., Shehory, O. and Sycara, K. 2000. Interleaving Planning and Execution in a Multiagent Team Planning Environment. *Technical report CMU-RI-TR-00-01, Robotics Institute*, Carnegie Mellon University, USA, February 5. Available from: http://www.ri.cmu.edu/pubs/pub_3274.html [Accessed 21 April 2008]

Reynolds, J. 2002. AI Game Programming Wisdom. In: Steve Rabin, editor. *Tactical Team AI Using a Command Hierarchy*. Charles River Media, Massachusetts, chapter 5.5, pp. 260-271.

Schubert, J., Wallén, M. and Walter, J. 2007. Morphological Refinement of Effect Based Planning. *Proceedings of the Third Int. Conf. Military Technology (MilTech3)*, June, Stockholm, Sweden, FOI – Totalförsvarets forskningsinstitut, ref Paper Or21.

Simple and Fast Multimedia Library 2008. Laurent Gomila. [Computer program] Available from: http://www.sfml-dev.org/ [Accessed 21 March 2008]

Smith, E. A. 2003. *Effect-Based Operations* [online]. The Command and Control Research Program. Available from: http://www.dodccrp.org/files/Smith_EBO.PDF [Accessed 28 April 2008]

Straatman, R., van der Sterren, W. and Beij, A. 2005. Killzone's AI: Dynamic Procedural Combat Tactics. *In Proceedings of the Game Developers Conference 2005*, San Francisco, CA, USA, March 7-11. Available from: http://www.cgf-ai.com/docs/straatman_remco_killzone_ai.pdf [Accessed 5 February 2008]

S.T.A.L.K.E.R.: Shadow of Chernobyl 2007. GSC Game World/THQ, Kiev, Ukraine. [Computer program]

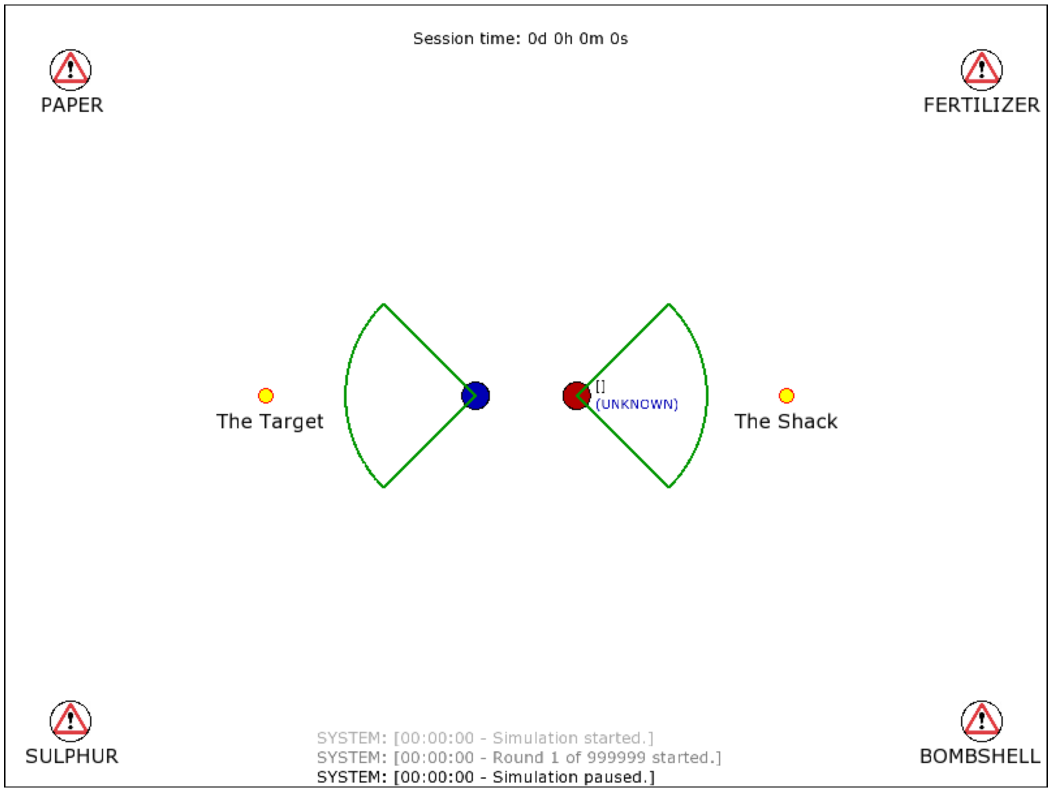
van der Sterren, W. 2002a. AI Game Programming Wisdom. In: Steve Rabin, editor. *Squad Tactics: Team AI and Emergent Maneuvers*. Charles River Media, Massachusetts, chapter 5.3, pp. 233-246.

van der Sterren, W. 2002b. AI Game Programming Wisdom. In: Steve Rabin, editor. *Squad Tactics: Planned Maneuvers*. Charles River Media, Massachusetts, chapter 5.4, pp. 247-259.

Wallace, N. 2004. AI Game Programming Wisdom 2. In: Steve Rabin, editor. *Hierarchical Planning in Dynamic Worlds*, Charles River Media, Massachussetts, chapter 3.5, pp. 229-236.

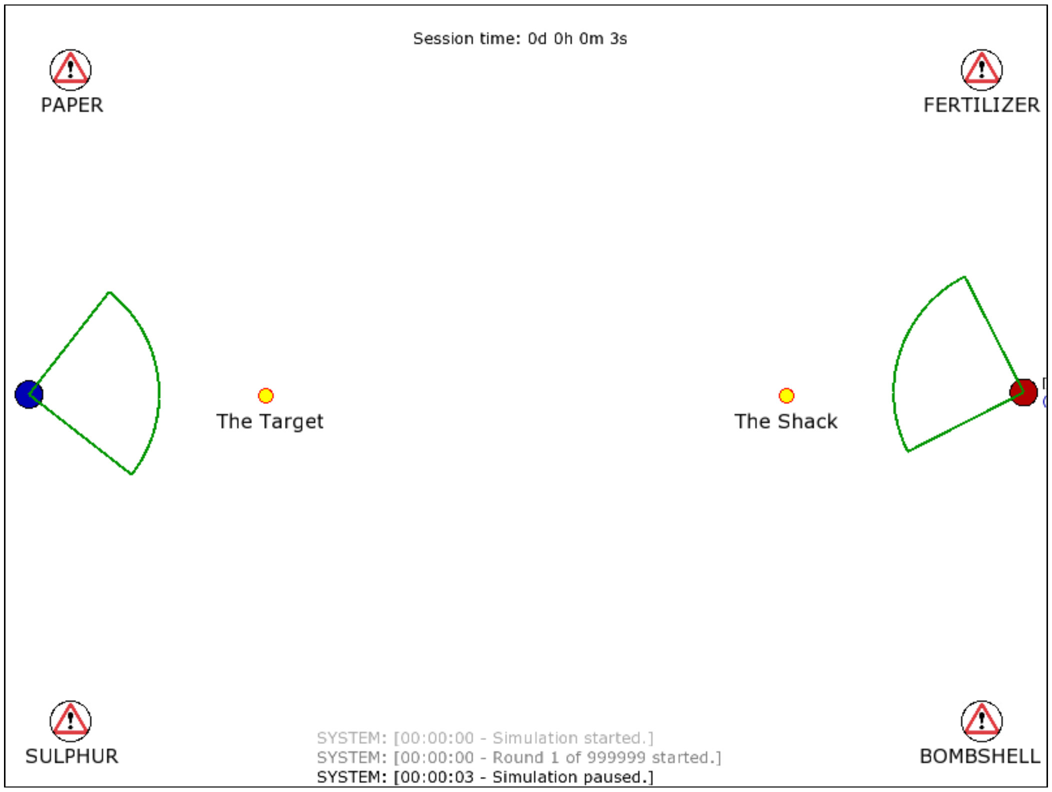
Appendix

The appendix holds the figures (screenshots) from the walk-through of section 6.2.2.



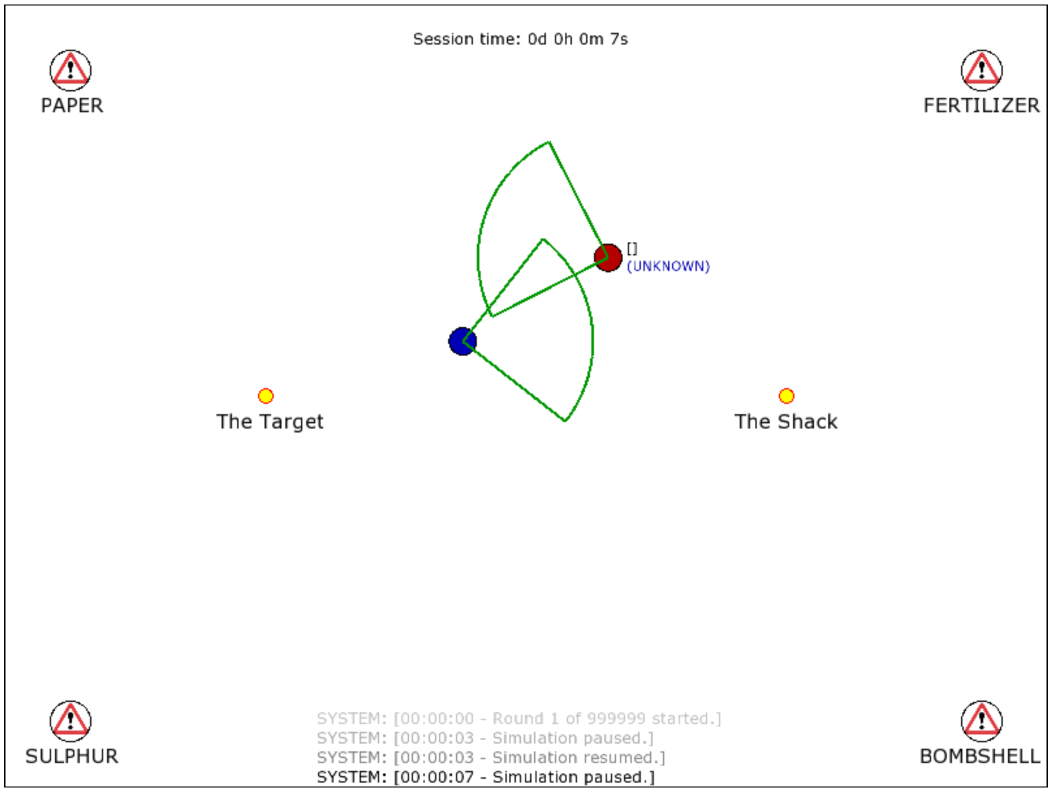
```
SYSTEM: [00:00:00 - Simulation started.]
SYSTEM: [00:00:00 - Round 1 of 999999 started.]
SYSTEM: [00:00:00 - Simulation paused.]
```

Figure 19. The simulation has just commenced and the two agents start facing away from each other, the LECOF agent being to the left.



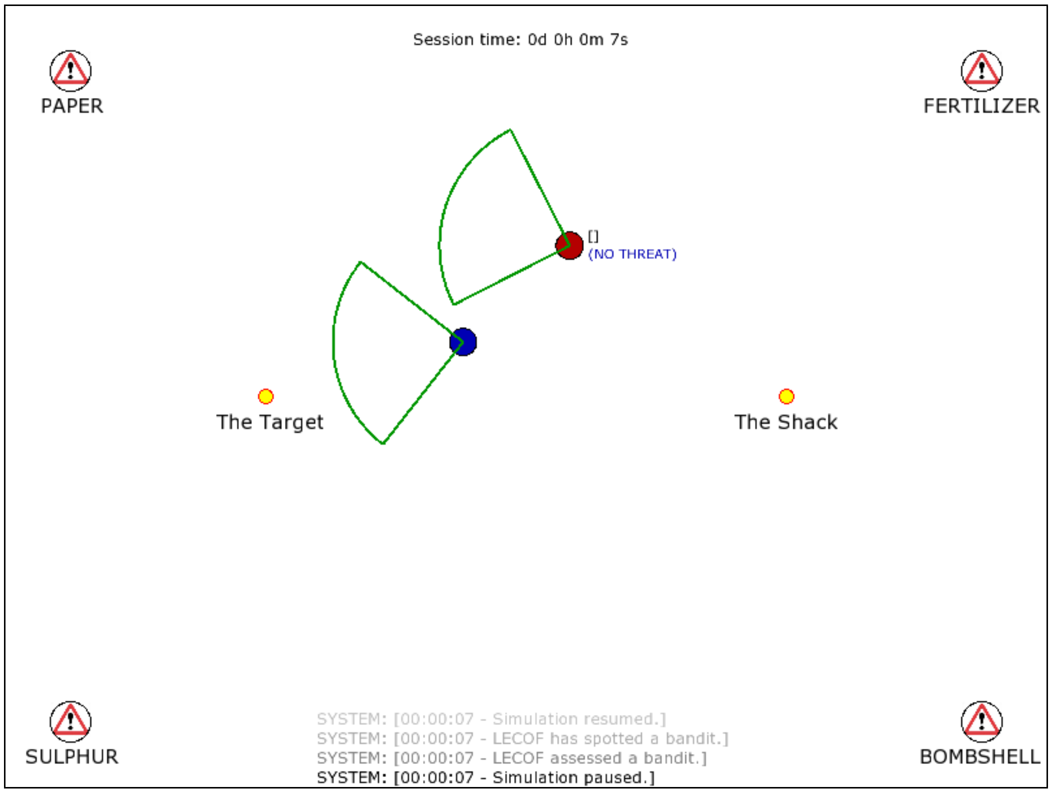
```
SYSTEM: [00:00:00 - Simulation started.]
SYSTEM: [00:00:00 - Round 1 of 999999 started.]
SYSTEM: [00:00:03 - Simulation paused.]
```

Figure 20. The agents have hit the world wall and have randomly selected new directions.



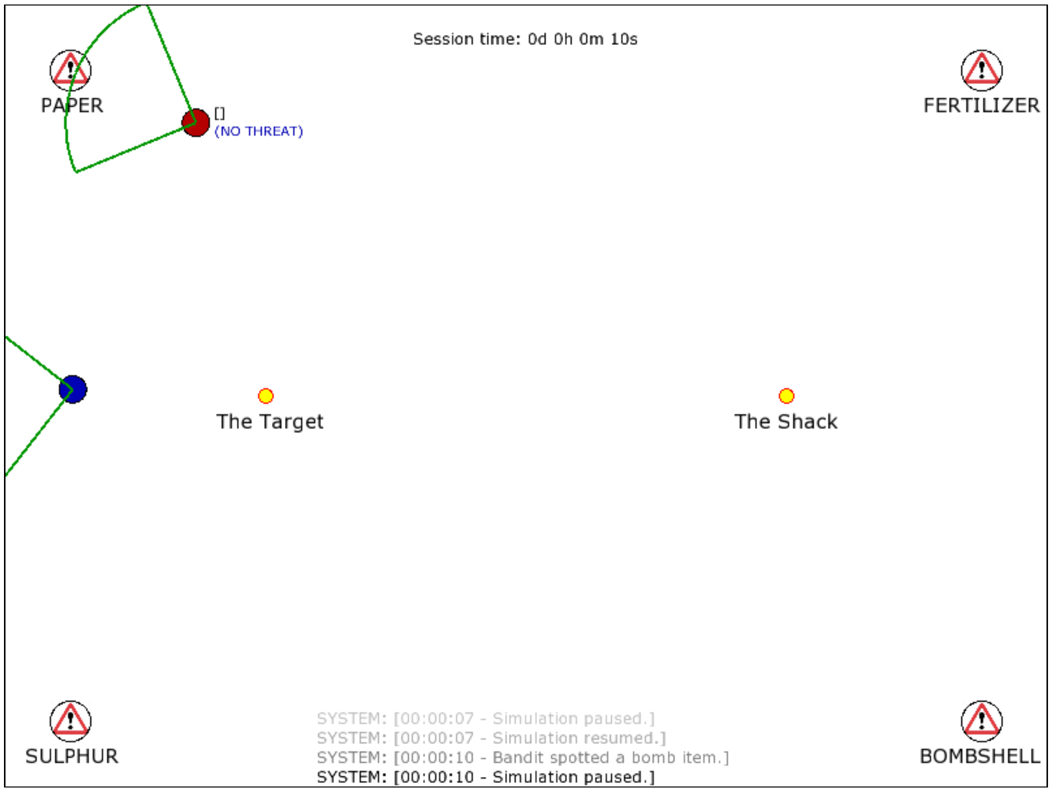
```
SYSTEM: [00:00:00 - Round 1 of 999999 started.]
SYSTEM: [00:00:03 - Simulation paused.]
SYSTEM: [00:00:03 - Simulation resumed.]
SYSTEM: [00:00:07 - Simulation paused.]
```

Figure 21. The Bandit agent is about to be assessed by the LECOF agent.



```
SYSTEM: [00:00:07 - Simulation resumed.]
SYSTEM: [00:00:07 - LECOF has spotted a bandit.]
SYSTEM: [00:00:07 - LECOF assessed a bandit.]
SYSTEM: [00:00:07 - Simulation paused.]
```

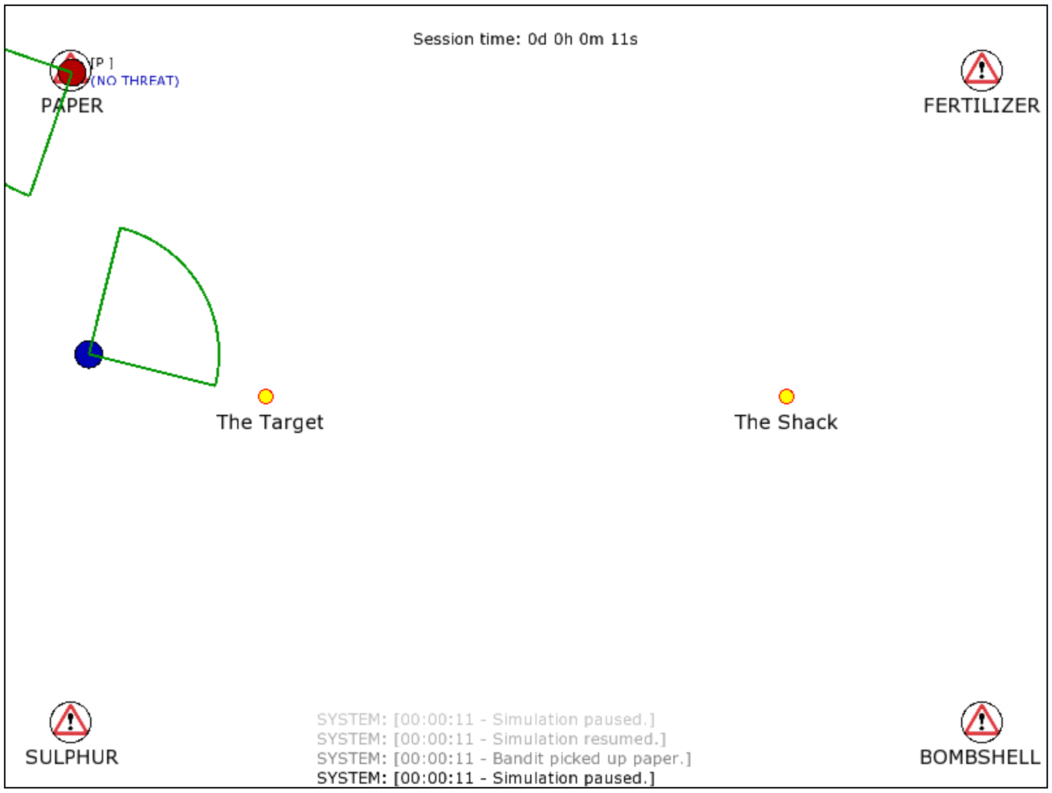
Figure 22. The Bandit agent has been assessed by the LECOF agent who deemed the Bandit of posing no threat.



```
Session time: 0d 0h 0m 10s
```

```
SYSTEM: [00:00:07 - Simulation paused.]
SYSTEM: [00:00:07 - Simulation resumed.]
SYSTEM: [00:00:10 - Bandit spotted a bomb item.]
SYSTEM: [00:00:10 - Simulation paused.]
```

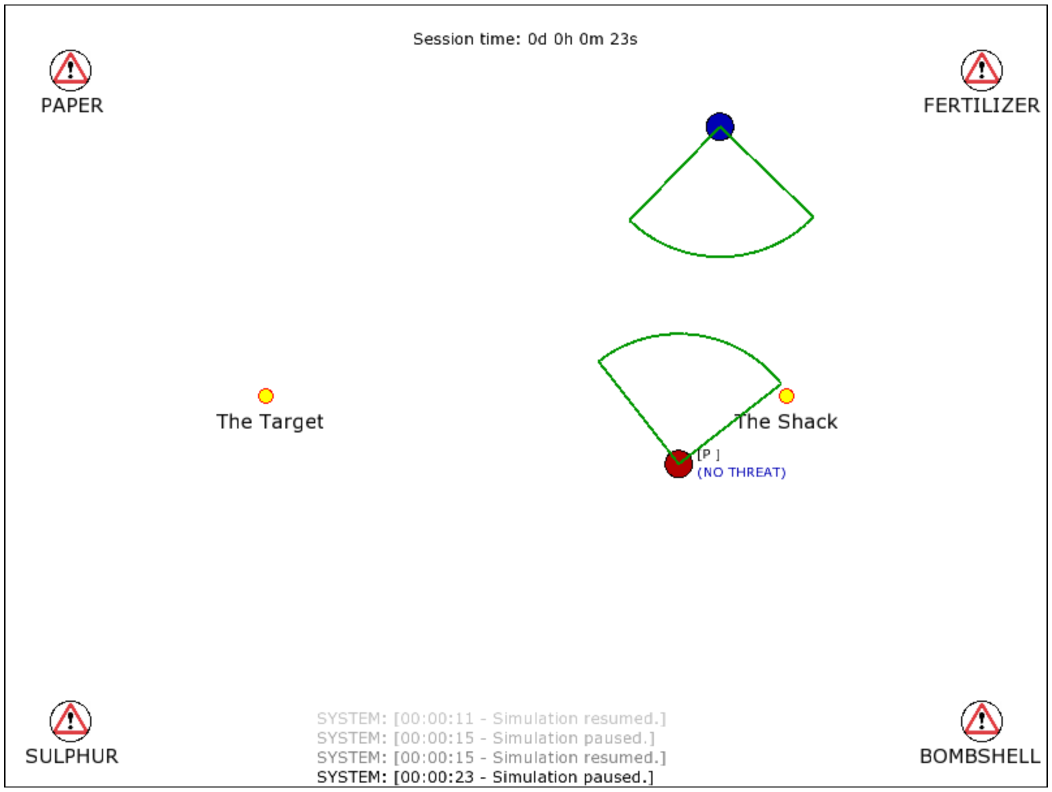
Figure 23. The Bandit agent spots the Paper item and changes action from `FindBombItem` to `PickUpBombItem`.



```
Session time: 0d 0h 0m 11s
```

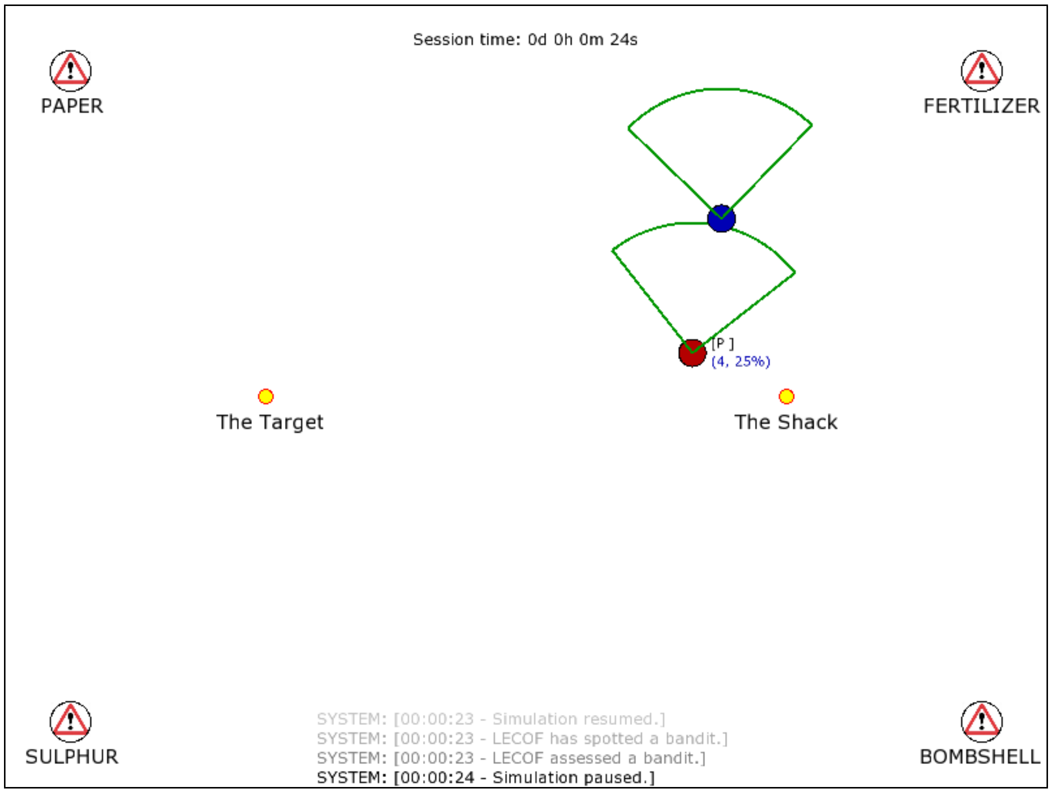
```
SYSTEM: [00:00:11 - Simulation paused.]
SYSTEM: [00:00:11 - Simulation resumed.]
SYSTEM: [00:00:11 - Bandit picked up paper.]
SYSTEM: [00:00:11 - Simulation paused.]
```

Figure 24. The Bandit agent has picked up the Paper item and starts once again roaming the world in search for bomb items.



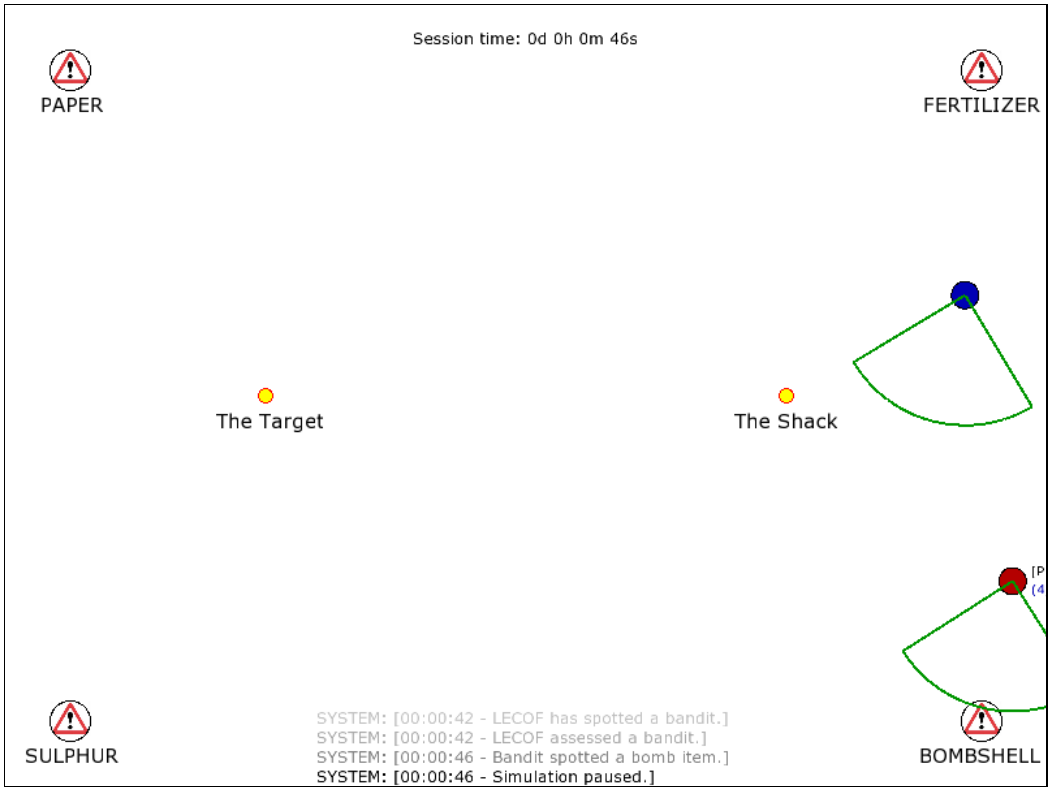
```
SYSTEM: [00:00:11 - Simulation resumed.]
SYSTEM: [00:00:15 - Simulation paused.]
SYSTEM: [00:00:15 - Simulation resumed.]
SYSTEM: [00:00:23 - Simulation paused.]
```

Figure 25. The LECOF agent is about to once again assess the Bandit agent.



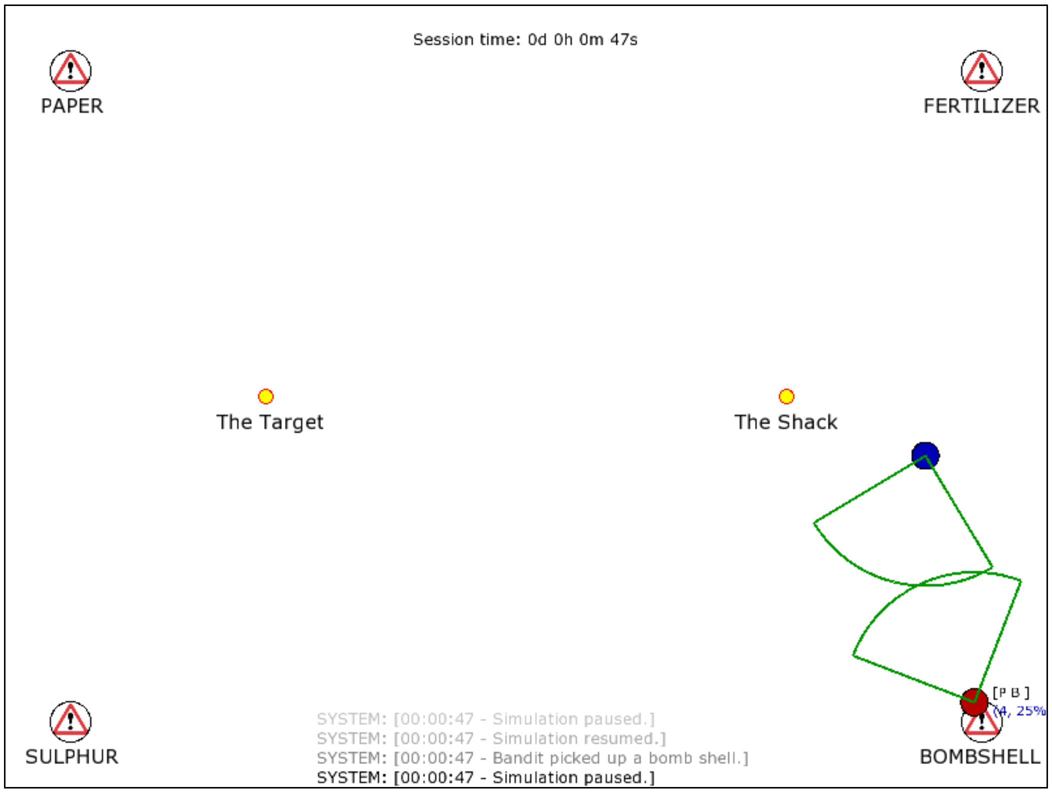
```
SYSTEM: [00:00:23 - Simulation resumed.]
SYSTEM: [00:00:23 - LECOF has spotted a bandit.]
SYSTEM: [00:00:23 - LECOF assessed a bandit.]
SYSTEM: [00:00:24 - Simulation paused.]
```

Figure 26. The LECOF agent has assessed the Bandit agent and determined it is of threat level 4 with 25 % completion.



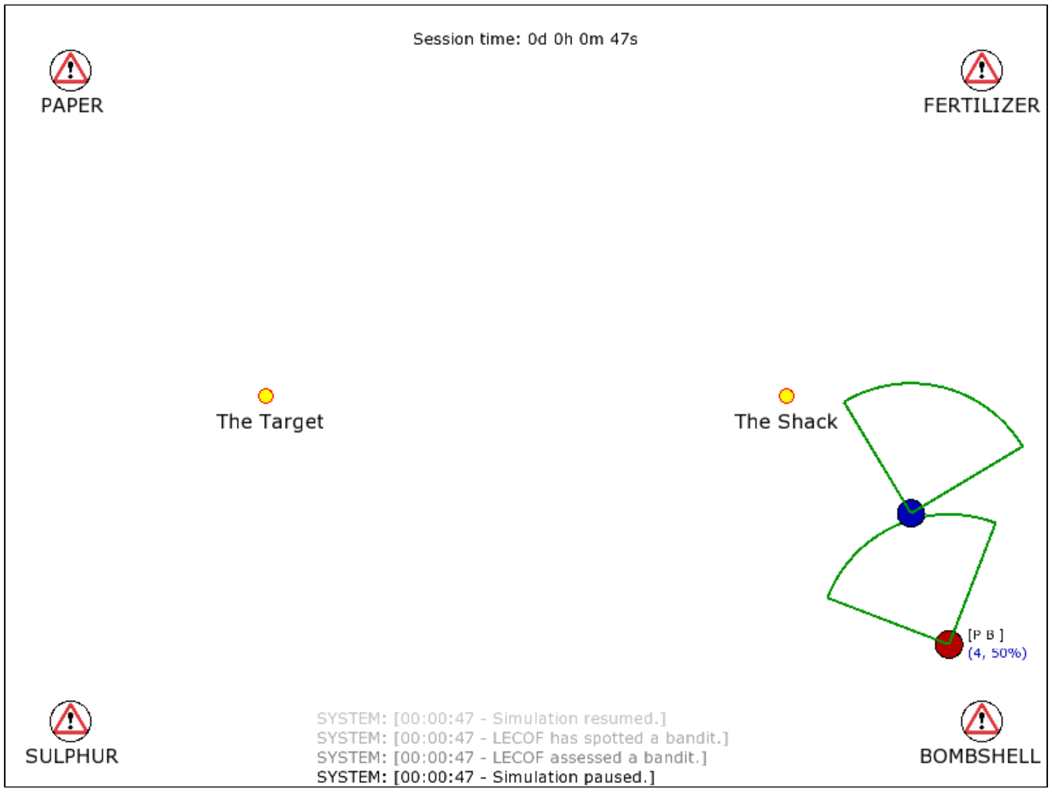
```
SYSTEM: [00:00:42 - LECOF has spotted a bandit.]
SYSTEM: [00:00:42 - LECOF assessed a bandit.]
SYSTEM: [00:00:46 - Bandit spotted a bomb item.]
SYSTEM: [00:00:46 - Simulation paused.]
```

Figure 27. The Bandit agent spots the bombshell item.



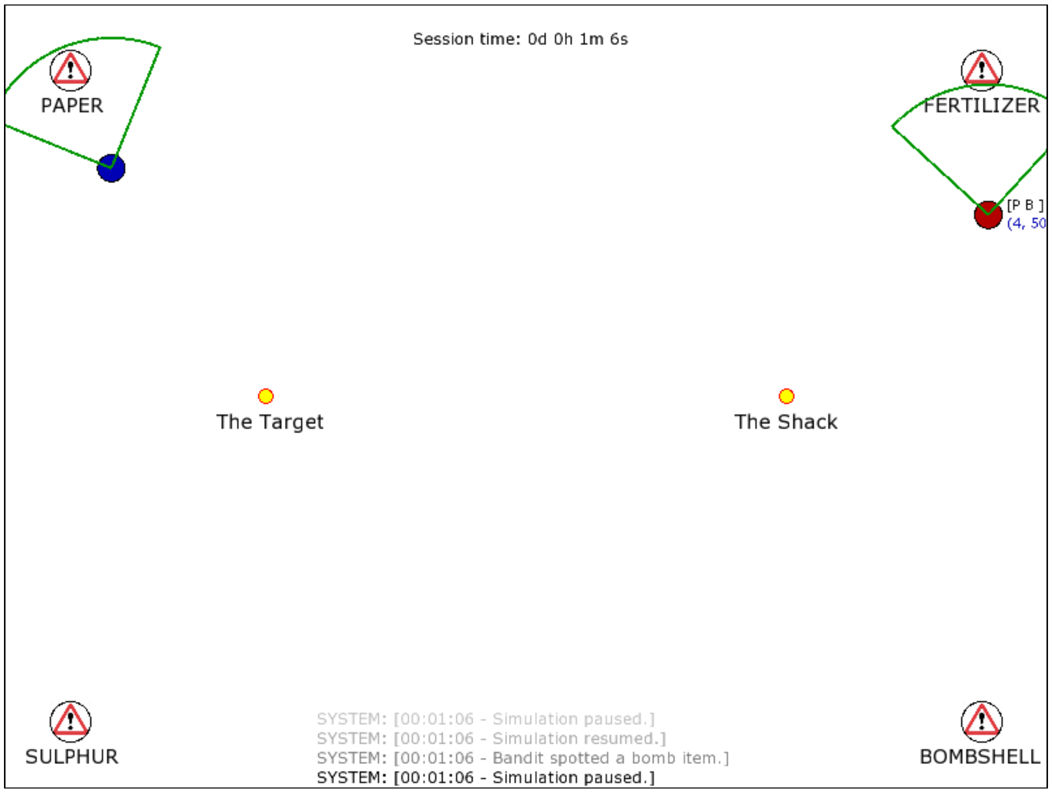
```
SYSTEM: [00:00:47 - Simulation paused.]
SYSTEM: [00:00:47 - Simulation resumed.]
SYSTEM: [00:00:47 - Bandit picked up a bomb shell.]
SYSTEM: [00:00:47 - Simulation paused.]
```

Figure 28. The Bandit agent picks up the bombshell item.



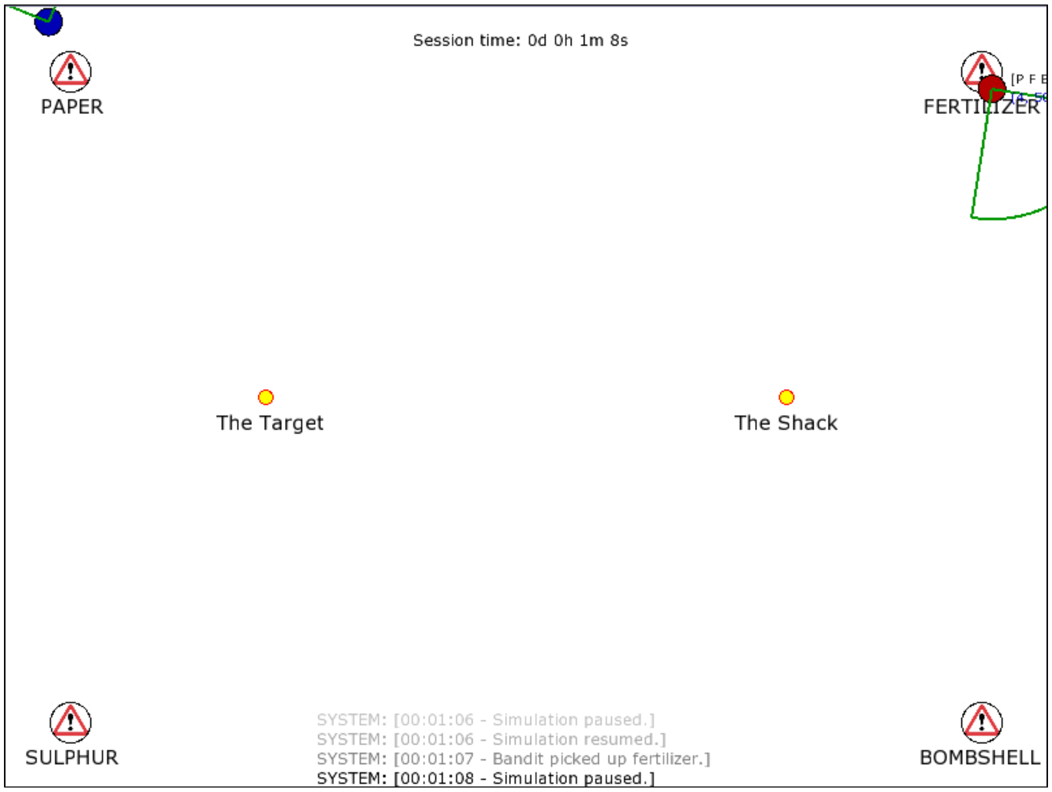
```
SYSTEM: [00:00:47 - Simulation resumed.]
SYSTEM: [00:00:47 - LECOF has spotted a bandit.]
SYSTEM: [00:00:47 - LECOF assessed a bandit.]
SYSTEM: [00:00:47 - Simulation paused.]
```

Figure 29. The LECOF agent has assessed the Bandit agent and tagged it with a level 4 threat with 50 % completion.



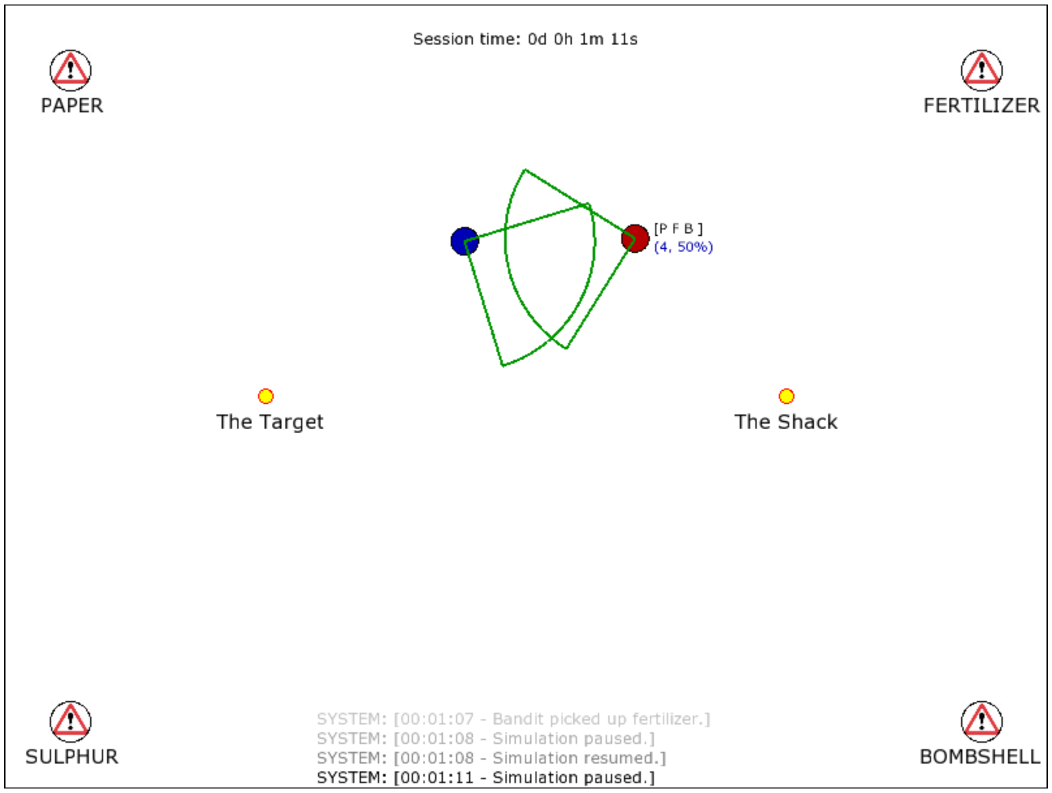
```
SYSTEM: [00:01:06 - Simulation paused.]
SYSTEM: [00:01:06 - Simulation resumed.]
SYSTEM: [00:01:06 - Bandit spotted a bomb item.]
SYSTEM: [00:01:06 - Simulation paused.]
```

Figure 30. The Bandit agent spots the Fertilizer item.



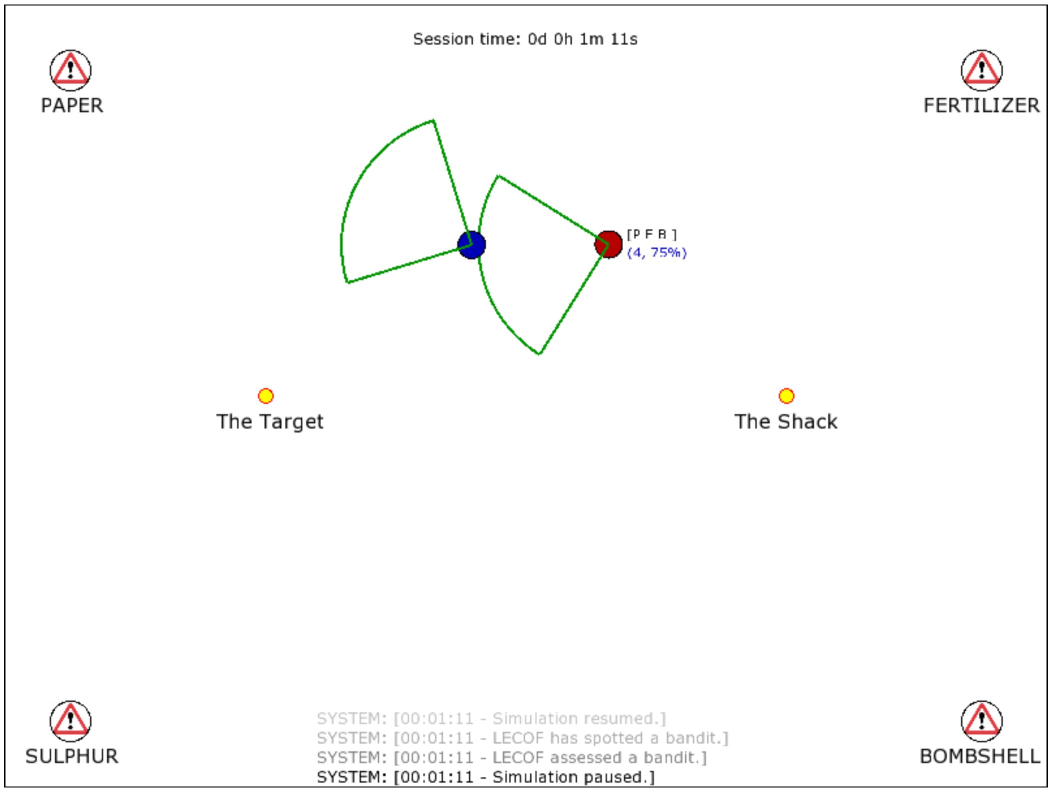
SYSTEM: [00:01:06 - Simulation paused.]
SYSTEM: [00:01:06 - Simulation resumed.]
SYSTEM: [00:01:07 - Bandit picked up fertilizer.]
SYSTEM: [00:01:08 - Simulation paused.]

Figure 31. The Bandit agent has picked up the Fertilizer item and is heading out to find the last bomb item.



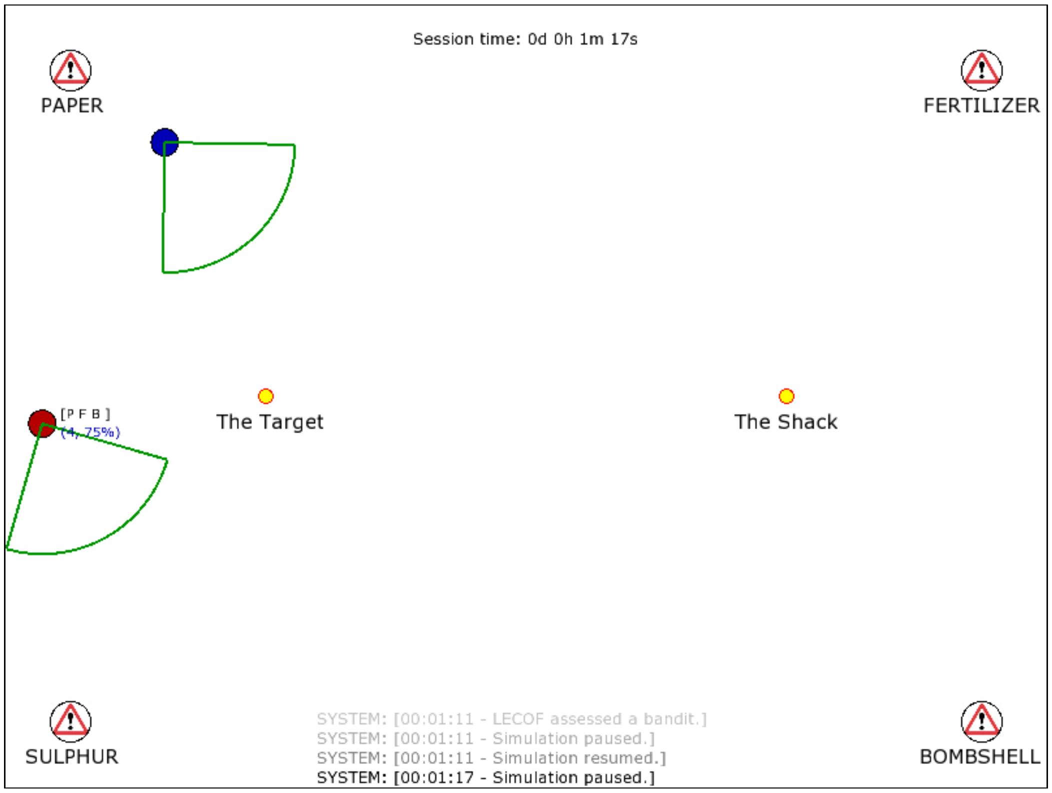
SYSTEM: [00:01:07 - Bandit picked up fertilizer.]
SYSTEM: [00:01:08 - Simulation paused.]
SYSTEM: [00:01:08 - Simulation resumed.]
SYSTEM: [00:01:11 - Simulation paused.]

Figure 32. The Bandit agent carrying the bomb items Paper, Fertilizer and Bombshell, is about to get assessed by the LECOF agent.



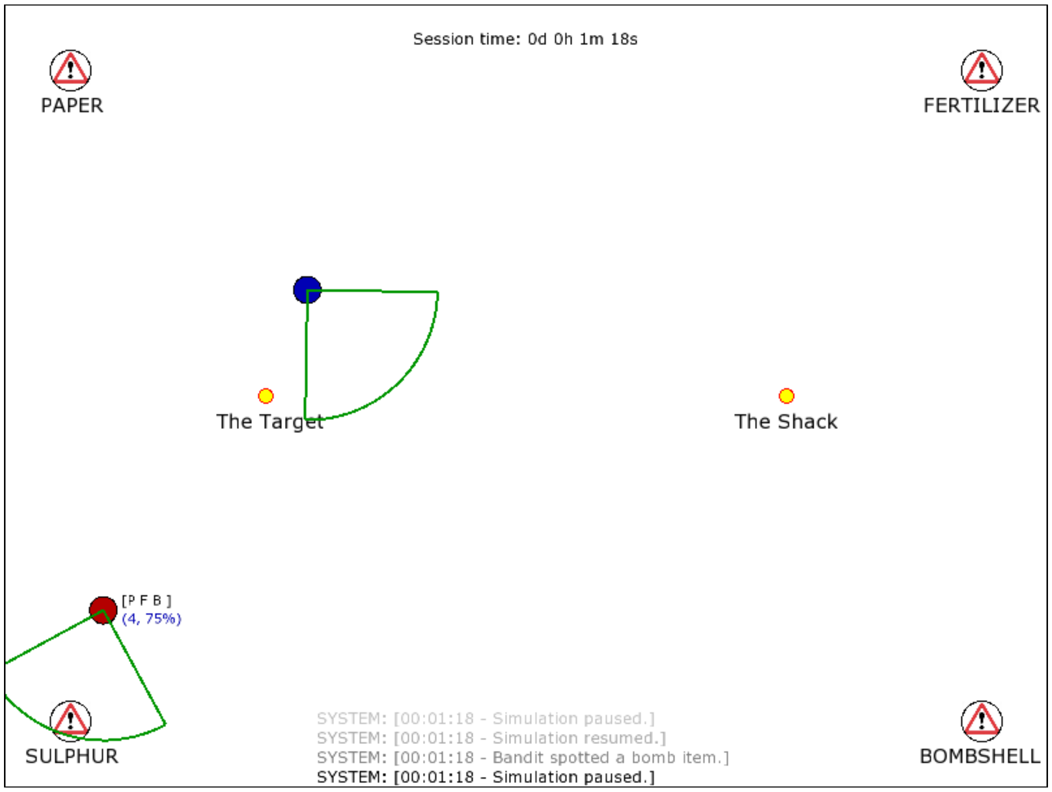
```
SYSTEM: [00:01:11 - Simulation resumed.]
SYSTEM: [00:01:11 - LECOF has spotted a bandit.]
SYSTEM: [00:01:11 - LECOF assessed a bandit.]
SYSTEM: [00:01:11 - Simulation paused.]
```

Figure 33. The Bandit agent has been assessed and the LECOF agent determined that the Bandit was of threat level 4 with 75 % completion.



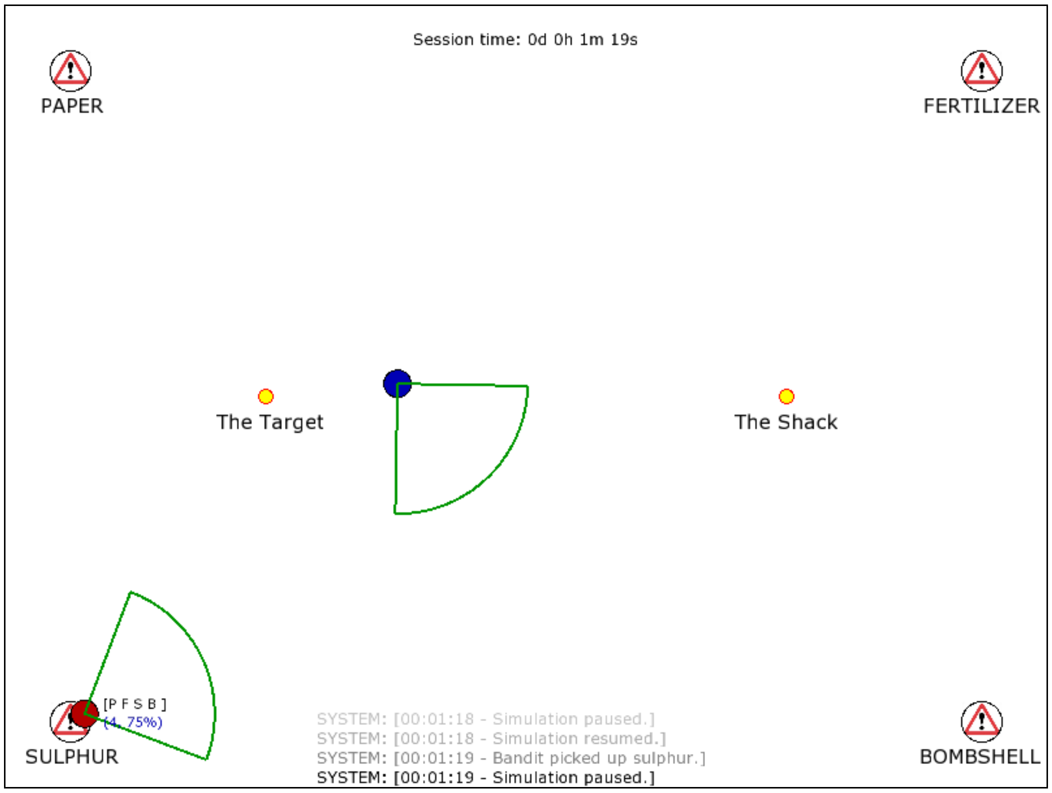
```
SYSTEM: [00:01:11 - LECOF assessed a bandit.]
SYSTEM: [00:01:11 - Simulation paused.]
SYSTEM: [00:01:11 - Simulation resumed.]
SYSTEM: [00:01:17 - Simulation paused.]
```

Figure 34. The Bandit agent has hit the wall and ends up going in the direction towards the Sulphur bomb item.



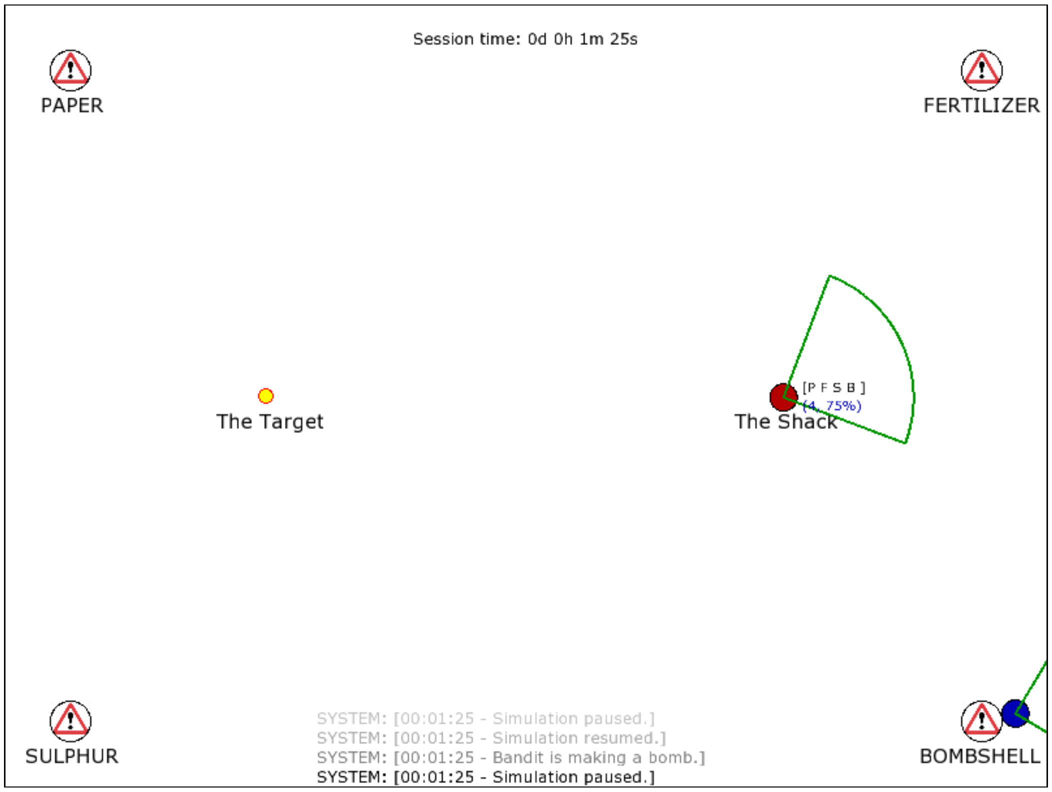
```
SYSTEM: [00:01:18 - Simulation paused.]
SYSTEM: [00:01:18 - Simulation resumed.]
SYSTEM: [00:01:18 - Bandit spotted a bomb item.]
SYSTEM: [00:01:18 - Simulation paused.]
```

Figure 35. The Bandit agent spots the Sulphur and heads towards it.



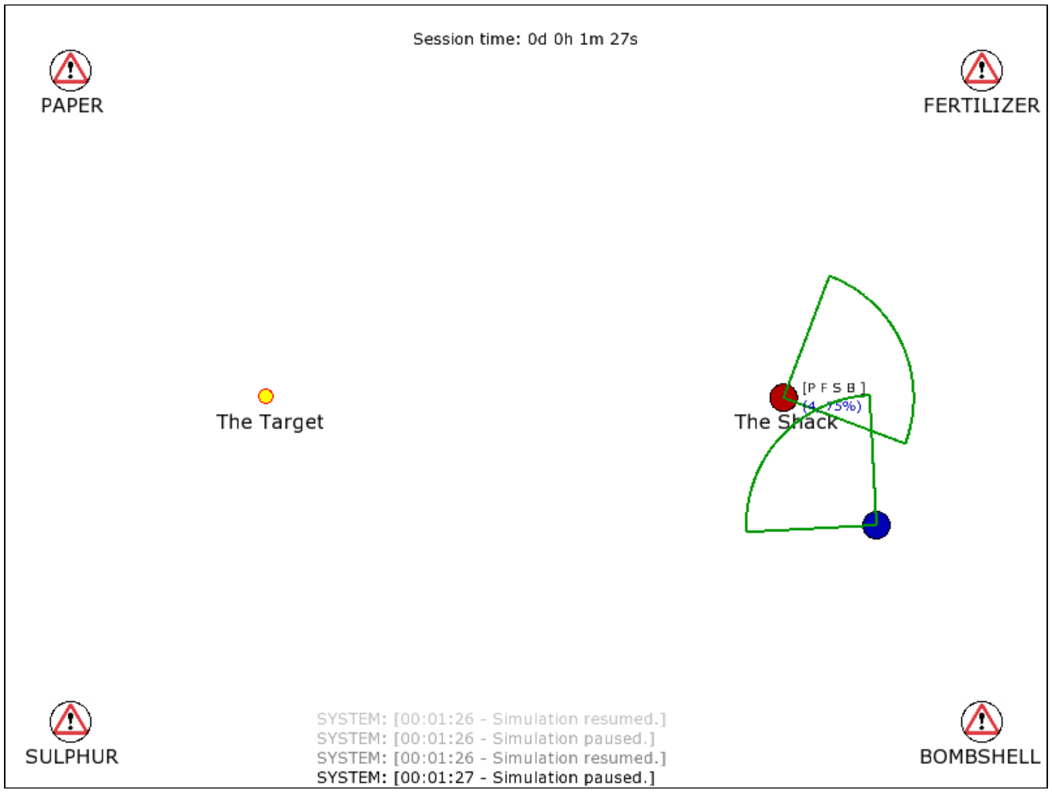
```
SYSTEM: [00:01:18 - Simulation paused.]
SYSTEM: [00:01:18 - Simulation resumed.]
SYSTEM: [00:01:19 - Bandit picked up sulphur.]
SYSTEM: [00:01:19 - Simulation paused.]
```

Figure 36. The Bandit agent has picked the last bomb item and sets off to make a bomb.



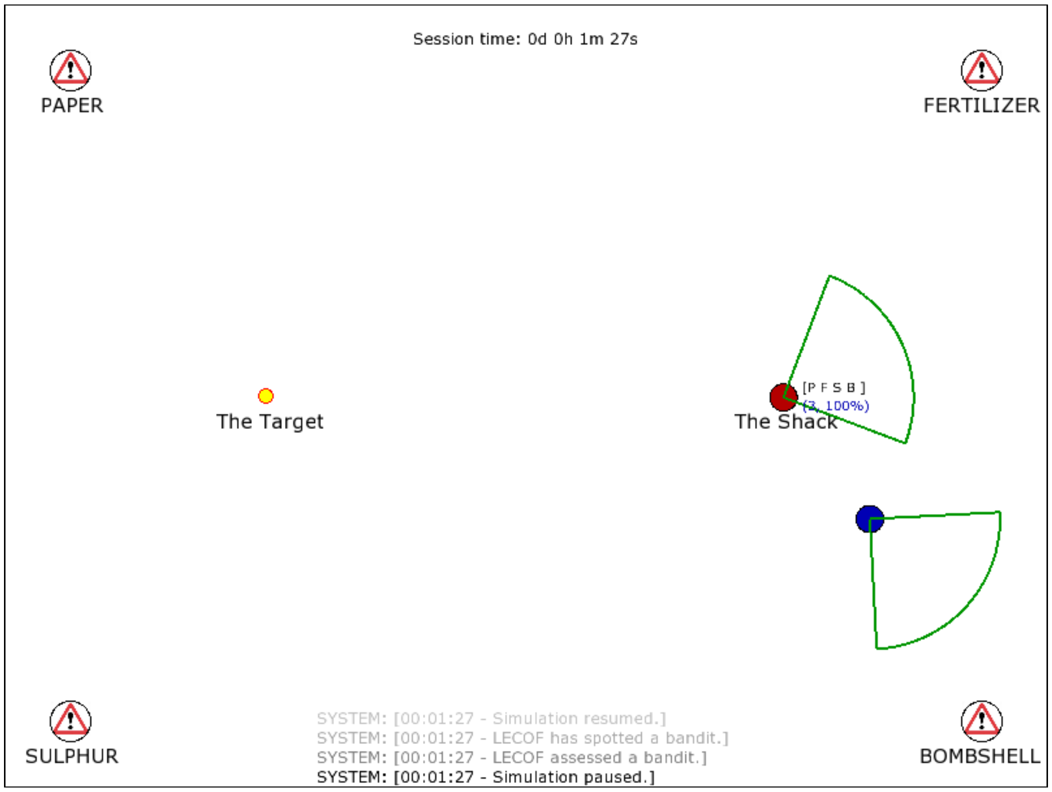
```
SYSTEM: [00:01:25 - Simulation paused.]
SYSTEM: [00:01:25 - Simulation resumed.]
SYSTEM: [00:01:25 - Bandit is making a bomb.]
SYSTEM: [00:01:25 - Simulation paused.]
```

Figure 37. The Bandit agent has arrived at The Shack, where the bomb items are to be turned into a bomb.



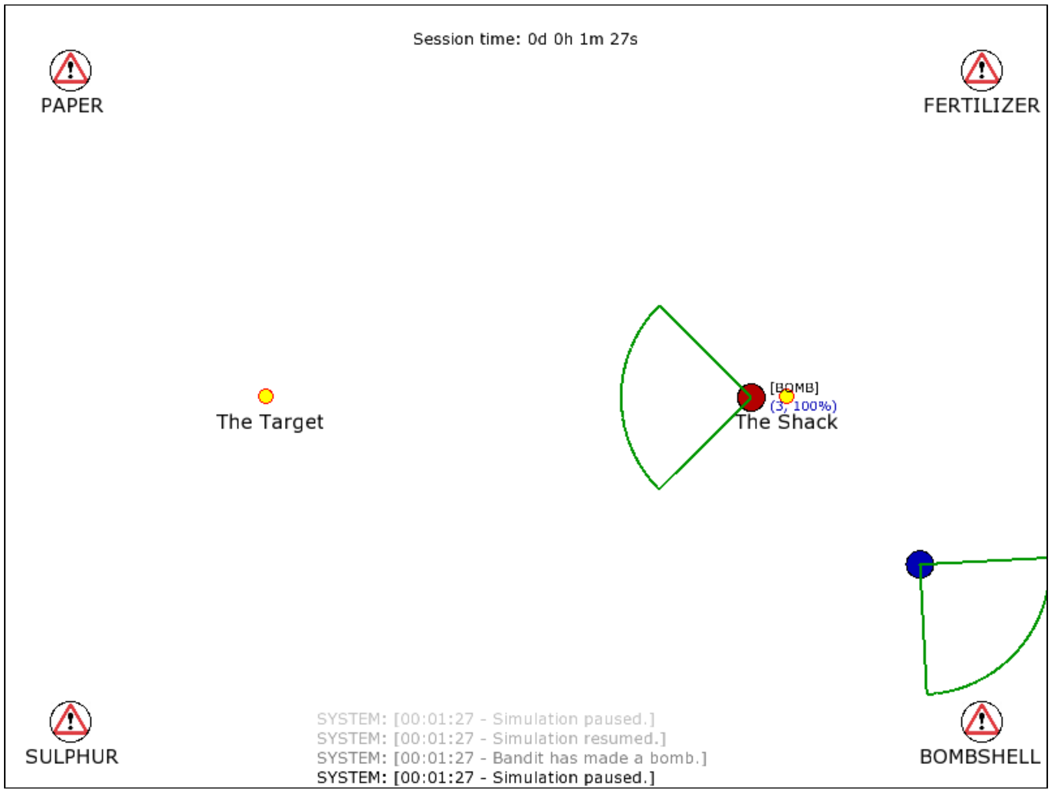
```
SYSTEM: [00:01:26 - Simulation resumed.]
SYSTEM: [00:01:26 - Simulation paused.]
SYSTEM: [00:01:26 - Simulation resumed.]
SYSTEM: [00:01:27 - Simulation paused.]
```

Figure 38. The LECOF agent is about to assess the Bandit agent.



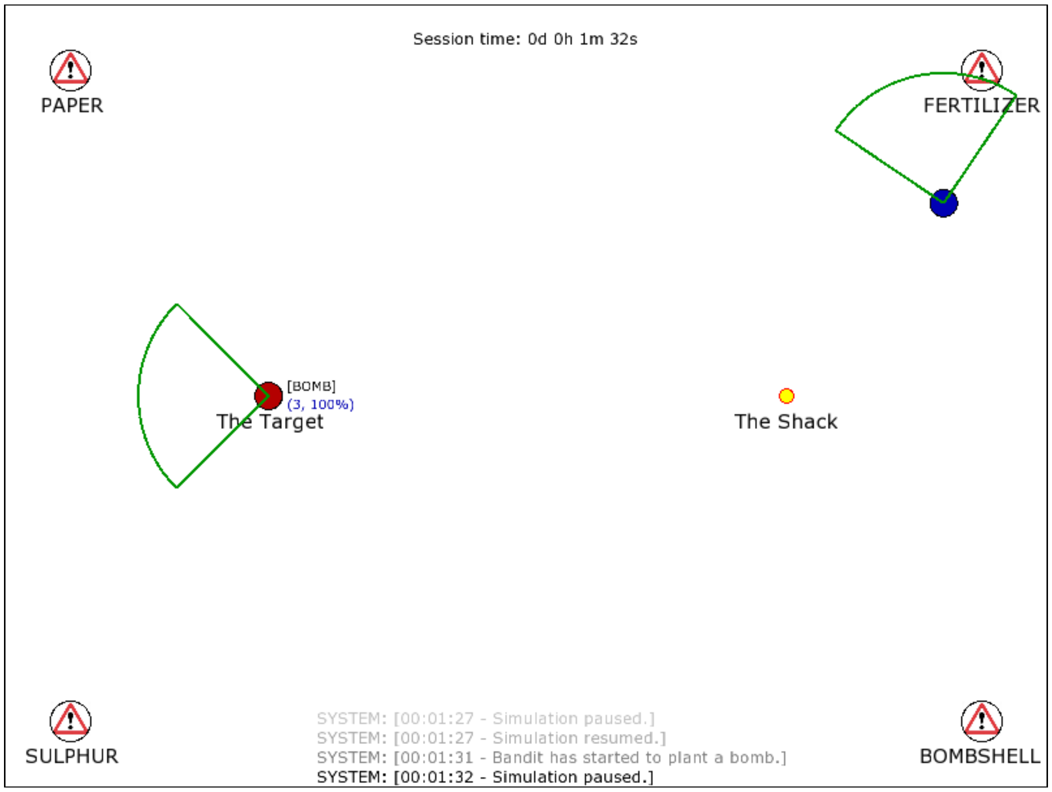
```
SYSTEM: [00:01:27 - Simulation resumed.]
SYSTEM: [00:01:27 - LECOF has spotted a bandit.]
SYSTEM: [00:01:27 - LECOF assessed a bandit.]
SYSTEM: [00:01:27 - Simulation paused.]
```

Figure 39. The LECOF agent has assessed the Bandit agent and determined it has threat level 3 with 100 % completion.



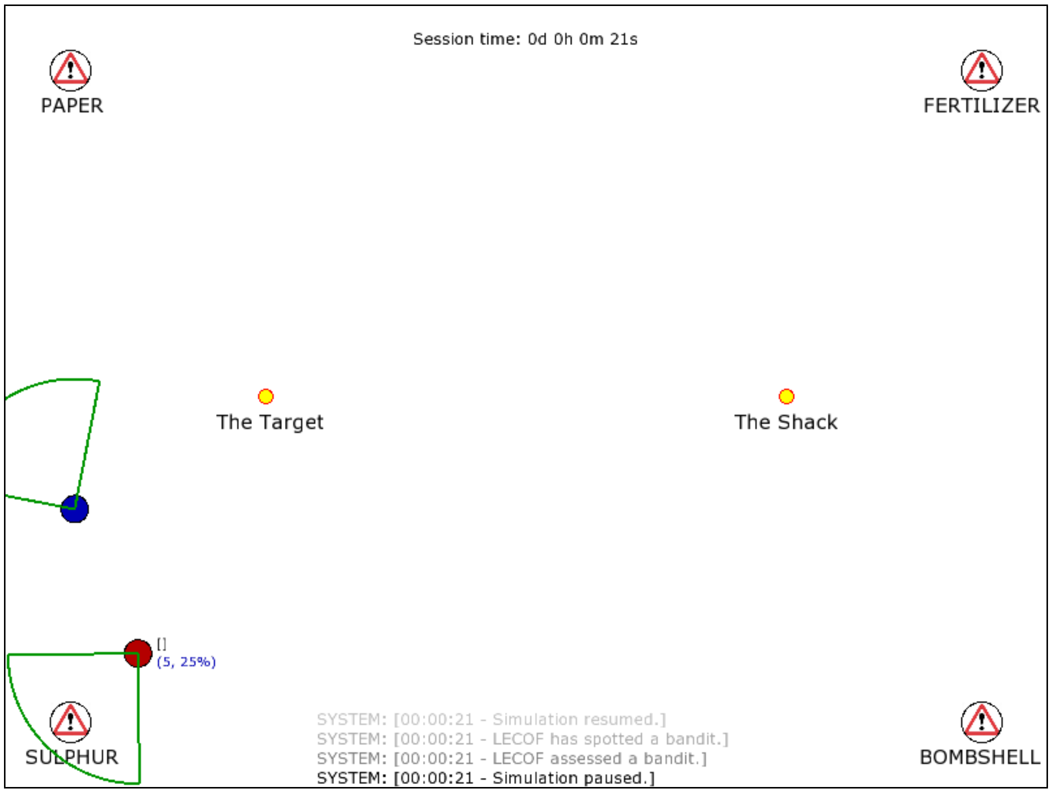
```
SYSTEM: [00:01:27 - Simulation paused.]
SYSTEM: [00:01:27 - Simulation resumed.]
SYSTEM: [00:01:27 - Bandit has made a bomb.]
SYSTEM: [00:01:27 - Simulation paused.]
```

Figure 40. The Bandit agent is done making the bomb and is heading towards The Target, where the bomb is to be planted.



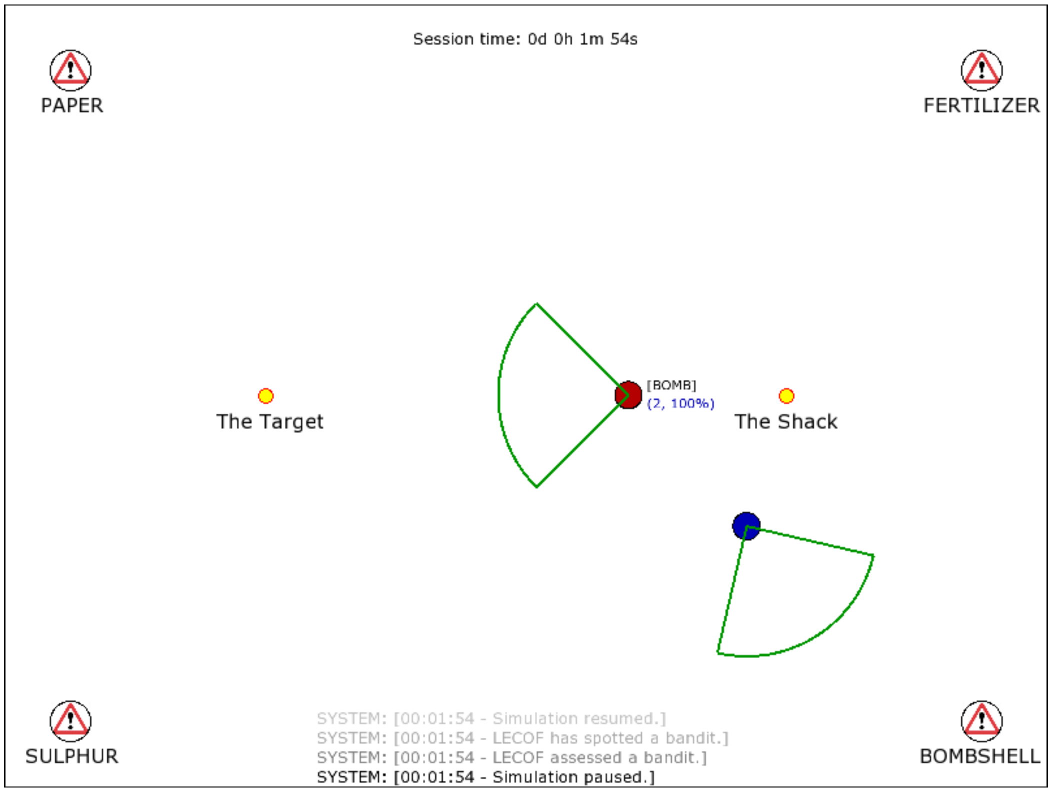
```
SYSTEM: [00:01:27 - Simulation paused.]
SYSTEM: [00:01:27 - Simulation resumed.]
SYSTEM: [00:01:31 - Bandit has started to plant a bomb.]
SYSTEM: [00:01:32 - Simulation paused.]
```

Figure 41. The Bandit agent plants the bomb at The Target, ending the round since the Bandit reaches its goal world state `(kTargetIsBlownUp, true)`.



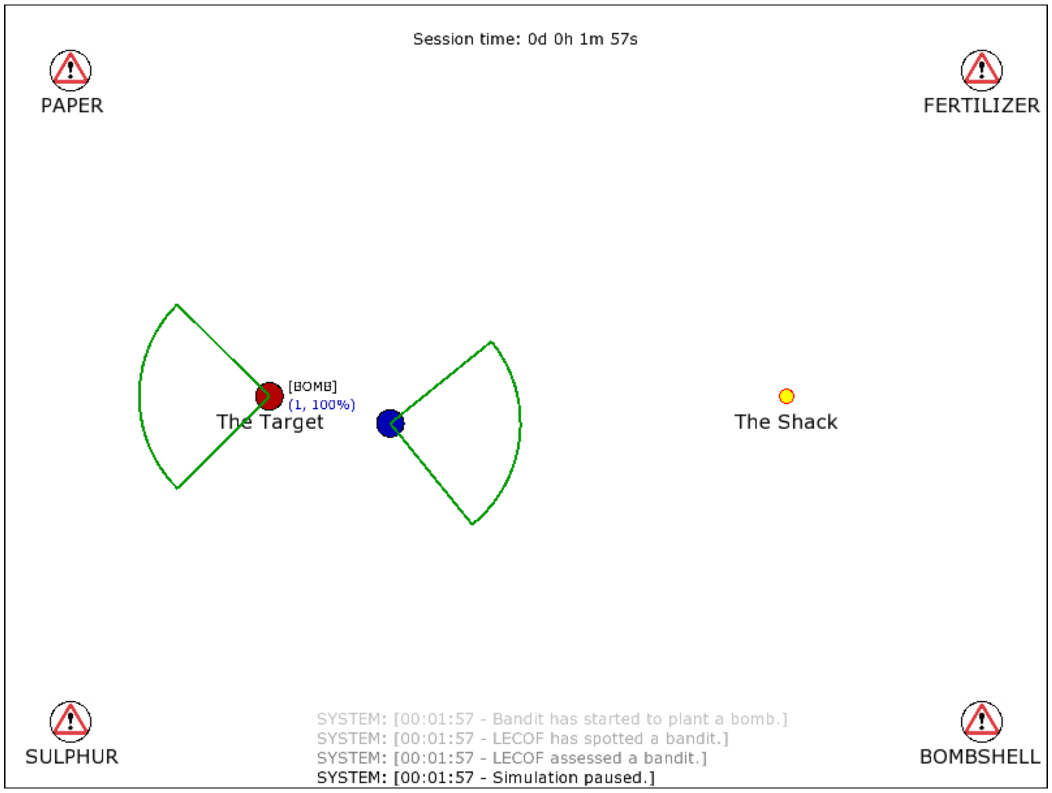
```
SYSTEM: [00:00:21 - Simulation resumed.]
SYSTEM: [00:00:21 - LECOF has spotted a bandit.]
SYSTEM: [00:00:21 - LECOF assessed a bandit.]
SYSTEM: [00:00:21 - Simulation paused.]
```

Figure 42. The Bandit agent, carrying no bomb items, has been assessed when spotting a bomb item, resulting in a threat level 5 tag with 25 % completion.



```
SYSTEM: [00:01:54 - Simulation resumed.]
SYSTEM: [00:01:54 - LECOF has spotted a bandit.]
SYSTEM: [00:01:54 - LECOF assessed a bandit.]
SYSTEM: [00:01:54 - Simulation paused.]
```

Figure 43. The Bandit agent has been assessed and found carrying a bomb, resulting in a threat level 2 tag with 100 % completion.



```
SYSTEM: [00:01:57 - Bandit has started to plant a bomb.]
SYSTEM: [00:01:57 - LECOF has spotted a bandit.]
SYSTEM: [00:01:57 - LECOF assessed a bandit.]
SYSTEM: [00:01:57 - Simulation paused.]
```

Figure 44. The Bandit agent has been assessed and found planting a bomb, resulting in a threat level 1 tag with 100 % completion.