A hierarchically-layered multiplayer bot system for a first-person shooter

By Tim Verweij, student number 1204955.
For the degree of Master of Science in Artificial Intelligence
with the specialization of Intelligent Internet Applications.
Completed August, 2007.

Internal supervisor
Dr. Martijn Schut
Artificial Intelligence Section
Department of Computer Science
Faculty Of Science
Vrije Universiteit of Amsterdam

External supervisor
Drs. Remco Straatman
Lead Programmer
Department of Artificial Intelligence
Guerrilla B.V.

Abstract

Each new generation of computer game consoles is characterized by increased computational power. Game developers are using this power to create games which are more beautiful and more realistic (often in a Hollywood movie kind of way) than their predecessors. Game AI developers have seen their memory and cpu budgets increase as well. The AI of first-person shooter games has increased in complexity, allowing for more entertaining and more life like behavior of the computer controlled enemies and allies. Although new AI systems bring new opportunities, they also come with new challenges. Previously successful, well known techniques may not be applicable anymore to new situations.

Computer game developer Guerrilla developed the first-person shooter Killzone (2004) for the PlayStation 2. Currently Guerrilla is working on the game's successor Killzone 2, which will be released on the PlayStation 3 platform sometime in 2008. Killzone 2 improves upon Killzone in many ways, including the AI. Where computer controlled soldiers in Killzone behaved according to fixed sequences of actions augmented by a behavior priority mechanism based on heuristic evaluations of the soldier's current context, Killzone 2 uses a specialized planner to construct these sequences dynamically, based on more detailed evaluation of the soldier's context by means of logical expressions.

Killzone included a number of multiplayer game modes, such as Assault and Domination. AI developers added relatively simple multiplayer bot AI for these game modes as well. Successor Killzone 2 also contains a multiplayer element with several game modes, some of which are similar to the ones found in Killzone. This thesis describes the results of an explorative study on Killzone 2's AI system. The study aims to answer how the Killzone 2 AI system may be used to create an extensible multiplayer bot architecture that supports the development of challenging and entertaining multiplayer bot AI which improves upon the AI of Killzone in the use of cooperative behavior.

We propose a hierarchically structured system to control the multiplayer bots, using one AI commander for each faction[1], each commanding several group leaders, each commanding several individual bots. We evaluate the system on the basis of a fully implemented AI for one of the multiplayer game modes.

Acknowledgements

This document is the final product of my time as intern researcher at Guerrilla, from January 2006 to August 2006. Part of this research was done in cooperation with Johan van der Beek, who also worked as an intern at Guerrilla during that time. As a result of this cooperation, the sections 1.1, 1.7 and the chapters 2 and 3 of this thesis were written together.

I would like to thank my internal supervisor Martijn Schut of the Vrije Universiteit of Amsterdam and my external supervisor Remco Straatman of Guerrilla for their support, ideas and constructive criticism, Johan van der Beek for his cooperation and humor, the other AI coders especially Arjen Beij and Robert Morcus for coding support, Eric Boltjes for teaching me some Maya, all participants of the multiplayer bot test rounds for their feedback and last but not least, I would like to thank my parents for their immense support.

[1] In the context of a first-person shooter, factions denote opposing groups. In Killzone, there are two factions: the Helghast and the ISA.

Contents

As stated in the acknowledgements, some sections of this thesis were written in collaboration with Johan van der Beek, who wrote his Master's thesis [1] during the same period. Johan van der Beek and Tim Verweij each wrote a part of the sections. The sections mentioned below appear identically (for the most part) in both theses. Below a specification of the division of the sections is given. All sections that are not mentioned below appear in this thesis only and are all written by Tim Verweij.
Sections written by Tim Verweij: 1.1.2 - 1.7 - 2 - 2.1.1 - 2.2 - 2.2.1 - 2.2.2 - 2.2.3 - 3 - 3.1 - 3.1.1 - 3.1.4 - 3.1.6 - 3.2 - 3.2.1 - 3.2.2
Sections written by Johan van der Beek: 1.1 - 1.1.1 - 1.1.3 - 1.1.4 - 2.1 - 2.1.2 - 2.1.3 - 2.1.4 - 2.3 - 2.3.1 - 2.3.2 - 3.1.2 - 3.1.3 - 3.1.5 - 3.2.3 - 3.2.4

1 Introduction

Each new generation of computer game consoles is characterized by increased computational power. Analogously, first-person shooter game software, including their AI systems have increased in complexity, allowing for more entertaining and more life like behavior of the computer controlled enemies and allies. Computer game developer Guerrilla is currently working on Killzone 2, which improves upon Killzone in many ways, including the AI. Killzone 2 uses a specialized planner to construct sequences of actions for each AI controlled soldier dynamically, based on a detailed evaluation of the soldier's context within the game.

Like Killzone, Killzone 2 includes a number of multiplayer game modes.. This thesis describes the results of an explorative study on Killzone 2's AI system. The study aims to answer how the Killzone 2 AI system may be used to create AI for multiplayer bots, making them entertaining and challenging, emphasizing on improved cooperative behavior compared to multiplayer bot AI of Killzone.

1.1 Contemporary FPS AI

First-person shooter (FPS) is a genre of computer games which is characterized by an on-screen view that simulates the in-game character's point of view and a focus on the use of handheld ranged weapons [7]. With each generation of FPS games, more powerful AI systems are designed and implemented to make the behavior of the Non-Player Characters[2] (NPCs) more diverse, lifelike and enjoyable for the player. To make NPCs in FPSs behave as realistic and human-like as possible, contemporary FPS AI developers have given each individual NPC a 'mind'. There is no global AI 'mind' controlling the individual decisions an NPC makes. However, AI that manages cooperation between NPCs might be global (See sections 2.1 and 2.2).

1.1.1 The purpose of a computer game

A computer game is not to be confused with a computer simulation. A computer simulation is an attempt to model a real-life or hypothetical situation on a computer so that it can be studied [35]. Therefore, the main focus of a simulation is to imitate reality as much as possible. The main goal of a computer game, however, is entirely different. Even though a game tries to pursue realism and world consistency, the game's primary purpose is to please the player that plays the game. Therefore, the player is the most important concept to be regarding when making a game. In FPS AI particularly, developers usually maintain a few rules of thumb regarding the realization of fun of the player. Below some of those rules are given [8].

Do not try to beat the player

While developers of AI chess computers continuously try to find methods in order to beat human chess players[3], beating players in FPS contributes to those players' frustration rather than the feeling of competition. When the AI constantly beats the player, whether the AI is cheating or not, the player regards the AI's game play as being unfair. While chess has strict rules to which the computer and the player have to comply to, the AI in a computer game can easily cheat and take advantage of information a player could never have, e.g. knowing the player's position on the game field even if the AI cannot visually perceive the player [8].

[2] The terms NPC, agent, soldier, individual, squad member or the AI are used interchangeably in this thesis.

[3] The defeat of the human chess master Kasparov by chess computer Deep Blue was seen as prestigious.

The AI must be predictable

The player must be able to understand the AI's decisions. The strategies the AI uses should be recognizable and predictable, so the player knows how to react to a certain strategy. Imagine an FPS where the AI would be smart enough to learn a new strategy every match. The player would never be able to defeat the AI, because the AI is unpredictable. When a player learns the strategy of the AI and figures out how to respond to that behavior, the AI changes its behavior and makes it impossible for the player to counter the AI's strategy. Unpredictability is frustrating for the player, because the player cannot determine how to react [8]. Hence, in commercial FPS AI, academic studies as Machine Learning [21], Evolutionary Methods [22] and Neural Networks [23] are generally considered less suitable for implementation than AI methods that produce predictable NPC behavior. Another reason than unpredictability is quality assurance. When using such system, the AI could become 'too smart'. For instance, if the AI gets the opportunity to prevent a player from finishing a level by blocking a door, so the player cannot advance to the next level, the game lacks quality. Another example is that at a certain point in time a friendly NPC has to tell plot critical information to the player, so the player knows the weakness of the final enemy of the game and is able to defeat him. However, the NPC has learned from previous games that some time after telling the plot critical information, he gets shot for telling this information to the player. The NPC decides that it is better not to tell the player the information in order to prevent himself from being shot, leading to the fact the player is unable to defeat the final boss.

Be sure what to show in front of the player

When the AI performs a maneuver, it should be performed in the player's field of view. When a maneuver is performed while the player is looking the other way, it makes no difference if the maneuver ever took place at all in the player's sense. Even if the outcome of the maneuver is still visible and persistent in the game world, the player might even be puzzled why the world changed in his absence. On the other hand, the AI should not stand in the player's line of sight blocking its view and line-of-fire to a possible threat [33].

1.1.2 Planning techniques

To bring the NPCs to life, AI developers have various tools at their disposal. With the arrival of faster computers, the techniques that are used to build the 'minds' of the NPCs have grown in complexity. Until some years ago, the AI of NPCs in FPS games consisted primarily of fixed sequences of actions (fixed plans) [30]. These sequences of actions are often implemented as a Finite State Machine (FSM), a tool to model behavior composed of states, transitions and actions [6] (See section 2.1.1). The behavior of NPCs may be controlled by several of these FSMs; each of them specialized in accomplishing a specific goal. On top of this palette of behaviors, a selection mechanism decides which of the available goals is most important, thereby selecting which of the available behaviors should become active.

A more recent development in FPS AI is the use of planning techniques (see sections 2.1 and 3.2) [30]. These techniques enable NPCs to construct a sequence of actions based on the context (e.g. presence of enemies) and their own state (e.g. how many bullets they have in their weapon) before executing this sequence. By planning ahead, NPCs can avoid starting a sequence of actions that cannot be completed. During the execution of a plan, NPCs may evaluate if the initial assumptions that started the plan are still valid. Since the planning approach decouples the planning of

actions from the execution of actions (as opposed to the mixed approach in FSMs), AI developers can add variations in behavior more easily.

1.1.3 Scripting versus autonomy

In game development, scripters have the task to create a story line for the single-player game environment. This story line is often static and takes the player through different situations.

The use of planning techniques allows the NPCs to compose their own behavior sequence, so it gives the NPCs more choices in what to do in a certain situation, i.e. the autonomy of the NPC has increased. With the increase of autonomy of NPCs, scripters are relieved of the task of scripting basic behavior when e.g. scripting a movement route between two locations or ordering the NPC to attack an enemy. Besides, the use of planning techniques allows the NPCs to compose behavior that is more complex than basic behavior. Autonomous NPCs do what seems best in a certain situation. However, the best solution is not always the best when taking the player into account. For example, when it is best according to an NPC to open a door to a room and open fire into that room, for game play reasons, it might be better to let the player open the door. In that case the player feels more important, because he triggered the fight and feels more in control of the situation. To ensure the player's fun, scripters want control over the story line, so scripters often want to be in control of the NPCs in the story. In these instances, the script can give orders to the NPCs, which get a higher priority than most autonomous behavior.

1.1.4 Cooperative behavior

An extension to planned individual behavior (see section 1.1.2) is cooperative behavior. Cooperative behavior emerges when multiple agents perform a task together. This task can be simple cooperative behavior, e.g. a medic tending a wounded ally, or the task can be complex, e.g. coordinating multiple agents to attack enemy units from different sides (i.e. flanking). In FPS AI, the concept 'squad' is used to define a group of agents performing cooperative behavior. In military doctrine a squad is a cooperative military unit consisting of four to nine members, depending on nationality of the army. A squad has a squad leader. In FPS games that support squads, squads can have a squad leader NPC who gives orders to the other squad members. Squads can have a virtual squad leader, which has no physical entity in the game, but combines all the squad members' perceptions and gives orders based upon reasoning with that information. Chapter 2 discusses these and several more squad coordination systems.

1.2 Scope

This section defines part of the scope of this research. Subsection 1.2.1 explains the concept of multiplayer games in the context of a first-person shooter. Subsection 1.2.2 explains the differences between first-person shooter AI for single player games and for multiplayer games. Subsection 1.2.3 discusses cooperative behavior with human players and why this feature was left outside the scope. Subsection 1.2.4 contains an explanation of the two multiplayer game modes that are the focus of this research.

1.2.1 Multiplayer games

Both Killzone and its successor include multiple multiplayer game modes. A multiplayer game is a game in which more than one person can participate at the same time. Players either compete against each other (individually or in teams) or cooperate to achieve a common goal such as defeating a computer controlled enemy. A multiplayer game mode is defined by a set of rules and regulations that specify game objectives, win / lose scenarios and conditions for scoring and ranking. Multiplayer games are played by connecting multiple computers together using a computer network.

For any game mode, points are awarded for killing opponents. However, many game modes are about accomplishing certain goals in addition to killing the enemy. To put the focus on the game goals, more points are awarded for accomplishing these than for making kills. The nature and number of these goals depend on the game mode and may also be different for the two teams. Game mode 'capture the flag' for example is a symmetric game mode where both teams must attack the other base to retrieve the enemy flag while guarding their own home base. Here the goals of the two teams are the same, unlike Killzone's game mode 'Assault'. In that game mode one team has to defend key objectives that the other team has to destroy.

1.2.2 Bot AI versus single player AI

The purpose of single player AI is essentially different from that of multiplayer bot AI. This is because NPCs found in a single player games play an entirely different role than bots do in multiplayer games. These differences greatly influence the way bots should play the game and will therefore be explained in more detail in this section.

Generally speaking, a multiplayer bot has the same rights as a player. Any action that a player may perform with his or her character is also available to multiplayer bots.

In single player games, the player advances from stage to stage. Whenever the player is killed, the game restarts at the beginning of that stage. While playing one of these stages, the player may face many enemies. The purpose of these enemies is to provide a certain degree of resistance to the player as he or she advances through the level. As soon as one of these enemies is killed, its task is over and it does not return to the game. The enemies must be weaker than the player; otherwise, chances are that the player will not be able to defeat them. This would make it impossible to advance in the game.

In contrast, multiplayer games are played on one stage only. As the name suggests, several players play one game simultaneously. Since these games are not centered on one player, they do not restart when one of the players dies. Instead, these games continue and the player that died will have the opportunity to rejoin the game. In this game type, all participants have the same role, therefore all players and all multiplayer bots have the same amount of 'health points'. Multiplayer bots may also use the same weapons as players and they may pick up and use items and weapons found during the game.

Bots not only have the same rights as players, they should also behave like normal players. In contrast to the artificial intelligence controlling the NPCs in single player game modes, bots need information that will help them to move around as far as the multiplayer level reaches. NPCs in single player game modes generally stick to a smaller area of operation within a level and do not pursue enemy targets beyond some predefined barriers or distance limits. Bots are expected to be able to operate on every spot on the map, just like players. Ideally, bots should know the ins and outs of the levels they play in, as experienced players will also learn these quickly.

Apart from this, there is also a need for bots to have some sort of world model in memory. This model should contain information on the (last known) locations of the various players and other bots. Without this, bots would have to cheat by requesting the actual position of invisible teammates and enemy targets or they would quickly lose track of their targets. Bots are always team players and will use this knowledge in their decision process to find the best course of action given the current situation and team objective. This means for example that bots may keep defending even though they are heavily outnumbered.

1.2.3 Adapting to human players

In multiplayer games, bots and players may be in the same team. Bots are not required to perform complex interactions with players. As a rule of thumb, bots may assume that players are going to attack and that bots will have to do the 'more boring' job of defense. No higher level reasoning about the intentions of players is required.

Since the bots may be playing on the same side as human players, there is an opportunity for the bots and human players to cooperate and coordinate their actions. For this project however, this feature was left outside the scope. The AI of the bots should not try to guess the plans or intentions of human players. This implies that bots do not adapt their own actions to fit the actions of human players.

Although bots that adapt their actions to human players may seem like a good idea at first, there are several reasons for choosing against this 'feature'. For a start, trying to guess what the plans or intentions of a human player are just by looking at its actions is a very complex task. It potentially requires a lot of (CPU) time, which may not be available. (Every part of the game system, including the AI, has a specific CPU budget) On top of that, depending on the skill of the human player, the observed behavior may either be the result of a genius plan or it may just be some random movement without any goal whatsoever. Even if the bots correctly derive the intentions of the human player, there are limited ways in which this information can be used to display smart looking behavior. If the player does not notice the behavior as a reaction to its own actions, all the effort was in vein. This brings us to the conclusion that the costs / risks are high, while the benefits are not, eliminating this feature from the project.

1.2.4 Domination & Capture and Hold

This section explains the rules of the game mode "Domination" found in Killzone, and the more or less equivalent "Capture and Hold" found in Killzone 2. Both are multiplayer game modes (see section 1.2.1) intended for network play. Both Killzone and Killzone 2 feature multiple game modes. Since Domination and Capture and Hold are the main focus of this research, only these are explained here.

<u>Domination</u>

Domination is a team based game where each team must fight to gain control of switches that have been placed around the battlefield. [19] The configuration for this game mode includes a time, points and spawn limit. The game ends when the timer expires, or the points limit or the spawn limit has been reached. The spawn counter is raised for each player that enters the game for the first time and for each player that reenters the game after being killed. Some points are awarded for killing enemy

players. 1 point is scored for each opposing team kill; -1 point for each suicide[4]; team kills count as a death for the dead character and -1 point for the killer. The focus of the game mode, however, is on the switches.

$$TotalScore = IndividualKills + SwitchesScore$$

To gain control over a switch, players must stand within the capture radius of the switch and must make sure that no enemy soldiers are within capture range of that same switch during the capture process. If a team succeeds in keeping away the enemy long enough, the switch will change ownership to the capturing team. A player capturing a switch will receive 5 points and 5 points are deducted if the switch is captured by an opponent. 1 point is earned by the team for retaining ownership of a switch for 15 seconds. The game ends immediately whenever one of the teams manages to capture all switches at the same time.

Figure 1: Top view of a Capture and Hold area located on top of a concrete platform. The capture radius is indicated by a circle.

Capture and Hold

The Capture and Hold game mode changes one main aspect of the game. The switches, now flagpoles, can be used as spawn locations. Whenever a Capture and Hold area has been captured, players of the faction that has control over the area may spawn in that area. Whenever an enemy soldier comes within range of the spawn area, the area becomes neutral and cannot be used as a spawn area. The scoring system is identical to domination. The total score of each faction consists of the score obtained by eliminating enemy units and the score obtained by capturing and holding mission objectives. In contrast to the Domination game mode, capturing all locations at once does not end the game.

1.3 Objective

The goal of this study is to deepen the understanding of the Killzone 2 AI system and its possibilities in creating multiplayer bot AI, compared to the AI of Killzone 1. More specifically, the study should answer the following question:

[4] A player can commit suicide by falling from a great height or by getting hit by his/her own explosives.

How may the Killzone 2 AI system be used to create an extensible multiplayer bot architecture that supports the development of entertaining multiplayer bot AI which is more challenging than the multiplayer bot AI of Killzone 1?

In answering this question, it should also answer some questions on the planner which is part of the Killzone 2 AI system. Are there any clear advantages in using the planner? Are there any parts of the multiplayer bot system that require planning, or perhaps parts that are better solved in a more traditional way?

1.4 Motivation

At the end of Killzone's development, multiplayer bots were added to the game. For each of the game modes, AI was written that allowed the bots to pursue the main goals of that game mode. Even though some of Killzone's multiplayer modes coincide with those of Killzone's successor, the code for these bots cannot be reused or converted easily, due to the changes in the AI system. Furthermore, since Killzone's bots were developed in very little time, their understanding of how the game's goals can best be reached is limited. The shortcomings of the bot AI can best be described as how single player AI soldiers would be playing multiplayer games. Coordination between individual bots of the same team was either completely missing, or very basic. They did not operate in groups and their behavior relied primarily on their own observations. This situation has formed the interest and motivation for the development of a new multiplayer bot system for Killzone 2 using the new AI techniques and improving upon the coordination between bots.

1.5 Deliverables

- A multiplayer bot system capable of enabling AI characters to play the various game modes.
- A single application of this system: a strategy for one of the game modes.

The first deliverable comprises the design and implementation of a generic structure within the game's AI system that enables AI developers to implement behavior for each multiplayer game mode. The second deliverable is an actual application of the first deliverable to show how it may be used to implement AI for one of the game modes. The game mode chosen for this project is called Capture and Hold (explained in section 1.2.4). In a nutshell, it is about securing predefined areas on the map. A more detailed description of this game mode can be found in section 1.2.4. This game mode was chosen for two reasons. The first reason is that Capture and Hold is a very tactical game mode; There are several opportunities and potential benefits in the use of cooperative behavior (attack / defense). The second reason is that Capture and Hold features immobile targets (the areas to be captured). This prevents the need for AI reasoning about moving targets and enables the focus to be on cooperative behavior.

1.6 Evaluation

- Extensible multiplayer bot architecture

  In this context, extensible means that it should be possible to extend the system to incorporate new AI for other game modes and strategies. Even though, for this project, only one game mode will have a working implementation of the AI, the design of the framework should support other game modes as well. This ensures a greater chance that the system will be used eventually.

- Improved strategy over Killzone's AI

  This requirement can be split into two sub requirements. Since the Killzone bot AI did not have much cooperative behavior, this leaves room for improvement. As a first requirement, the new bot AI system should address this issue. The second requirement is that the new AI should outperform the old AI. Killzone included a game mode called domination, very similar to the Capture and Hold in the new game. The AI code for this game mode should be translated to the new system and tested against the new system.

- Entertaining bots

  This is perhaps the most important requirement of Artificial Intelligence for any computer game: it should make the game fun to play. Especially for FPS games, the AI is responsible for a substantial part of the overall game experience; therefore the AI should be entertaining.

- Challenging for novice players

  This requirement is connected to the previous one and to the expected use of FPS bots. Bots are often used to 'fill up a game' and are never expected to be as good as human players. However, bots that constantly lose are no fun at all; it is not rewarding to destroy weak opponents. On the other hand, bots should not be too hard to defeat either. The skill of the bots should be somewhere in the middle; challenging for novice players.

1.7 Guerrilla

Guerrilla is a game development studio, based in the heart of Amsterdam, the Netherlands. It was founded at the beginning of 2000 as the result of a merger between three smaller Dutch game developers, the company now employs 130 developers, designers and artists, encompassing 20 different nationalities. The first game released by Guerrilla, Shellshock: Nam '67 was developed for PC, Xbox and Playstation 2 and published by Eidos Interactive. In 2004, Guerrilla signed an exclusive deal with Sony Computer Entertainment. Under that deal, Guerrilla developed games exclusively for Sony's consoles[5]. After the release of Killzone for PlayStation 2 (2004), the company was acquired by Sony Computer Entertainment in 2005. It went on to release Killzone: Liberation for PlayStation Portable (2006), and is currently working on a new Killzone title for PlayStation 3: Killzone 2 [9], [10].

The Killzone game series is about warfare in a futuristic battle zone. In its games, Guerrilla strives for a certain degree of realism. Although in Killzone the battles take place on another planet, there are no laser or plasma weapons. A sentence taken from the back of the Killzone cover illustrates that the developers are "drawing inspiration from the best-known 'real war' scenarios of our time." Guerrilla also aims for realistic and believable NPC behavior. (From the Killzone cover: "Adversaries react like proper military units") In Killzone 2, Guerrilla aims to implement realistic tactical individual and squad behavior as long as this behavior is recognizable and believable for the player. Recognizable and believable behavior provides consistency throughout the game on which the player can trust, making the game fun to play. Realistic tactical behavior implies the use of military tactics and techniques.

[5] Guerrilla developed titles for Sony's game consoles Playstation 2 (PS2), Playstation 3 (PS3) and the Playstation Portable (PSP).

2 Related work

This chapter describes AI techniques used in other (non-Guerrilla titles) FPS games and simulations. Section 2.1 discusses techniques that were applied successfully in commercial games. Section 2.2 explains additional techniques that were only tried in a scientific and / or experimental setup. In addition, section 2.2 discusses literature about techniques that were not applied to FPS games, but are nevertheless useful when designing AI for such games. Finally, section 2.3 summarizes the decision and coordination systems that are discussed in this chapter. Where applicable, the descriptions focus on the workings of the high-level decision system for a single NPC as well as on the system coordinating the behavior of several cooperating NPCs.

2.1 Techniques used in applications

The applied AI in the commercial games discussed in this section can be divided into two parts. The decision system describes the AI structure and abilities of the individual NPCs in the game, while the coordination system explains the structure and the capabilities of coordinated behavior (if existent) in the game.

2.1.1 Quake III Arena

Quake 3 Arena is a multiplayer first-person shooter released on December 2, 1999, developed by Id Software. Quake 3 Arena is the third title in the series and differs from the previous games in the Quake series in that it excludes the normal single-player element, instead focusing upon multiplayer action. The solo experience in Quake 3 is arena combat versus AI opponents [32].

Decision system

For the Quake 3 bot [43], the main behavioral routines are structured as a Finite State Machine (FSM). An FSM is a model of behavior composed of a finite number of states, transitions between those states, and actions [6]. Quake 3 uses a procedural finite state machine, in which each state is coupled to a procedure capable of producing a specific behavior, specialized in dealing with a specific situation or task. To accomplish this, each state has a knowledge base of production rules (if-then-else rules).

At any moment in time, the bot is in one specific state. The production rules of the active state determine the behavior of the Quake bot[6]. These rules include conditions for state transitions. Whenever such conditions are met, the current state is disabled and the next state is activated. The newly activated state then takes over control of the behavior. Each state may have multiple conditions to transition to any number of other states. Figure 2 provides an overview of the FSM, representing procedural states as nodes and state transitions as arrows.

[6] A bot is an AI character that is designed for multiplayer games.

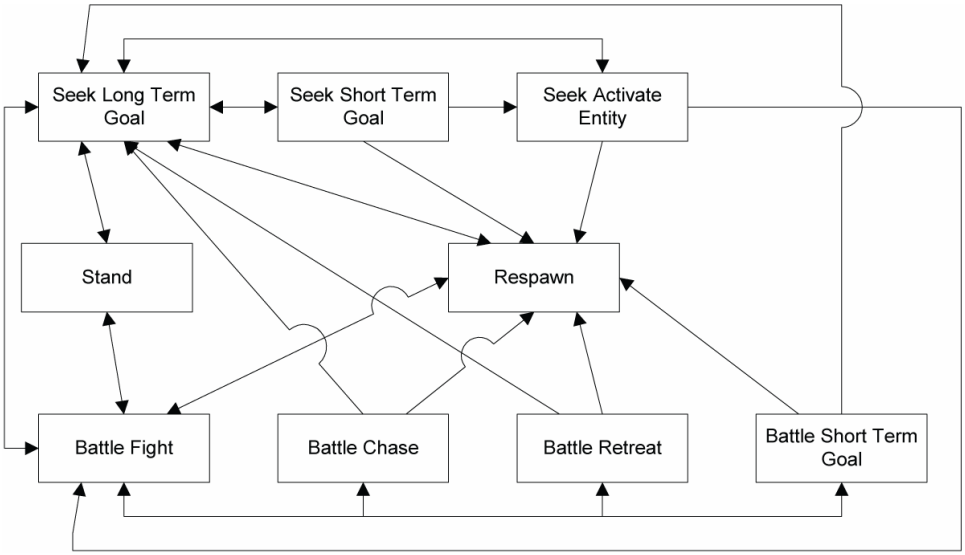

Figure 2: Overview of the Quake 3 bot FSM, representing procedural states as nodes and state transitions as arrows. The Battle Chase node not only determines how the bot behaves when it is trying to chase an enemy, but also when the bot stops chasing and starts fighting.

Coordination system

Quake 3 includes several multiplayer game modes that have special team objectives, e.g. Capture the Flag[7]. The Quake 3 bot is capable of participating in these games as it has actions specialized in dealing with these game modes. The activation of these actions however, is done on an individual basis and it will not take the actions of other team mates into account. To coordinate the individuals, in each team one of the members is selected as leader. By sending chat messages, the team leader may order team mates to attack the enemy or to capture their flag. The Quake bot has a special module that will allow it to be the team leader as well. For Capture and Hold[8] it will balance the number of attackers and defenders depending on the state of the game. Besides sending orders to teammates, it will also assign itself a task.

2.1.2 NOLF 2

No One Lives Forever 2: A Spy In HARM's Way (NOLF 2) was released by Monolith Productions in 2002. NOLF 2 is a humorous FPS spy game set in 1960 in which the female protagonist has to spy on the terrorist organization HARM. Missions in NOLF 2 can only be completed by fighting, sneaking, using gadgets, or a combination of those [27].

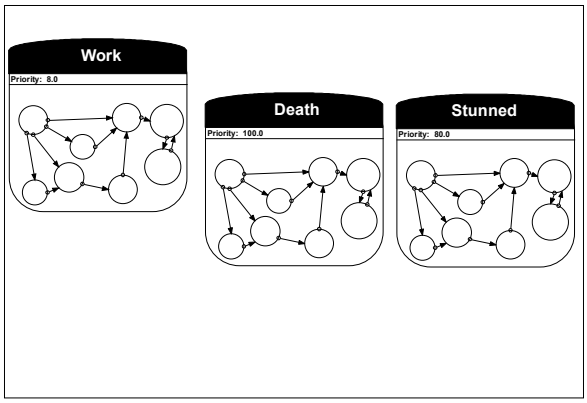

Figure 3: NOLF 2's goals as boxes with embedded plans

Decision system

NOLF 2 uses goal directed autonomous agents. Agents in NOLF 2 constantly select the most relevant goal to control

[7] Capture the Flag is a game-mode, in which the players and bots have to take the flag from the base of the enemy and return it to their own base.

[8] Capture and Hold is a game-mode in which the team gets points for controlling and defending specific areas in the level.

their behavior. The active goal determines the agent's behavior through an embedded plan, a hard coded sequence of state transitions, so each plan is an FSM. When the agent has a certain goal, the agent sequentially adopts the states of the embedded plan to reach that goal. In the embedded plan there can be different ways (branches) to reach a goal depending on the preconditions of each way, but those branches are predetermined at the time the goal is coded. Figure 3 shows the goals as boxes in which the fixed embedded plans are schematically drawn [28].

As all goals in NOLF 2 are stand-alone, a problem with the hard coded FSM in NOLF 2 is that goals cannot communicate with each other. Therefore, each goal has to be fulfilled completely in order to have the agent in the default position from which he can adopt a new goal. For instance, when the agent is working at his desk and gets shot by another agent, the agent will put away his work, stand up from the desk, push his chair under the desk and finally fall to the ground dead. [30]

Coordination system

To coordinate multiple agents NOLF 2 uses a Blackboard System. A Blackboard System is a shared object that allows inter-agent communication through posting and querying public information [15]. The blackboard in NOLF 2 allows agents to post records, remove records, query records, and count records. A record consists of the ID of the posting agent, the ID of the target object and some record information.

In FPS games, it is often the case that multiple agents are shooting at a threat. In many of these cases the line-of-fire from an agent to a threat can be blocked by an ally. The blackboard of NOLF 2 provides a solution to this problem. Generally, when an agent finds himself obstructed by an ally, the agent can do four things:

- The obstructed agent can do nothing.
- The obstructed agent can fire anyway, possibly killing his ally.
- The obstructed agent can move.
- The obstructed agent's ally can move.

The first two options are not optimal and often make the agent look stupid. The other two options need some form of communication, which in NOLF 2 is provided by the blackboard system. The obstructed agent first determines that his line-of-fire is blocked. The next step is to determine whether the obstructing object is an ally. The obstructed agent can then now ask his ally to duck, so the agent can fire over his head. When the ally denies the request the agent can move to another position. To make this cooperation process look even more intelligent, the obstructing agent can shout at the ally to duck as part of the requesting process [29].

2.1.3 FEAR

First Encounter Assault Recon (FEAR), a first-person shooter developed by Monolith Productions and published by Vivendi, was released in 2005. FEAR is an FPS in a horror setting in which the protagonist works for the secret special operations group FEAR of the US government dealing with paranormal threats [5].

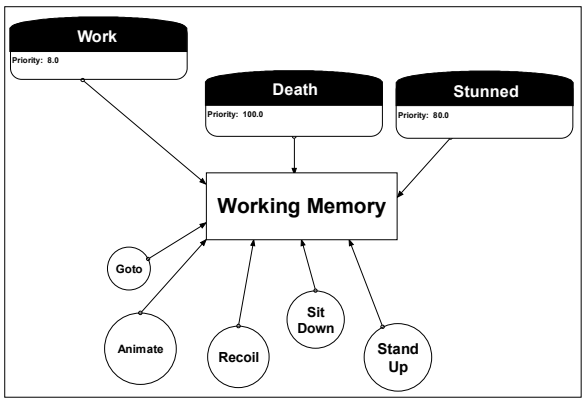

Figure 4: The decoupled goals and actions of F.E.A.R.

Decision system

FEAR uses Goal-Oriented Action Planning (GOAP) as decision system. GOAP defines the conditions that have to be met in order to satisfy a certain goal. As in NOLF 2, the agents in FEAR constantly select the most relevant goal to control their behavior. At every point in time when goal selection takes place, goals compete with each other for selection. When a goal wins this competition, that goal is considered best for the situation the agent is in, thus that goal is selected. After the selection the agent supplies the goal to GOAP after which GOAP generates a plan to satisfy that goal. Unlike the embedded plans in NOLF 2, goals and actions are decoupled (see Figure 4). GOAP has all the actions the agent can perform available and GOAP can predict the outcome of an action. For instance, the outcome of the action `reloadWeapon` is a reloaded weapon at the end of the action. GOAP tries to find a path among all available actions the agent can perform in order to find such a sequence of actions that the goal is satisfied. This path is called a plan. [30]

Coordination system

The coordinated behavior in FEAR can be divided into simple and complex squad behaviors. Simple behaviors involve laying suppression fire, sending the AI to different positions, or have the AI follow each other. Complex behaviors handle things that require more detailed analysis of the situation, like flanking[9], coordinated strikes, retreats, and calling for and integrating reinforcements. Complex behaviors are not actually programmed in FEAR, but emerge from simple squad behaviors. For instance, the squad goal `get_to_cover` can emerge into a flanking maneuver when a squad member (accidentally) gets into cover at a flanking position. [30]
Simple squad behavior in FEAR consists of four basic behaviors:

- `get_to_cover` has the squad members to lay suppression fire as the squad members not in cover get into cover.
- `advance_cover` squad members lay suppression fire as the squad members move closer to the threat.
- `orderly_advance` moves the squad in a single line to another position having each squad member cover the squad member in front of him and having the last squad member covering the rear.
- `search` splits the squad into pairs and has the squad systematically search rooms in a certain area.

Simple squad behaviors follow four steps. Squad AI tries to find squad members to participate in the squad behavior. If the squad AI finds participants, the squad behavior activates and the squad AI sends orders to the participants. The squad member receives the orders as a goal. That goal will compete for priority among the other goals the squad member has as normal.

2.1.4 Halo 2

Halo 2, an FPS developed by Bungie Studios, was released for the Xbox game console in 2004. Halo 2 features a science fiction setting in which the protagonist has to prevent a collective of genocidal alien races from destroying earth and its inhabitants [11].

[9] Flanking an enemy is attacking the enemy from multiple sides at the same time

Decision system

Halo 2 uses a Hierarchical Finite State Machine (HFSM) as decision system. As mentioned in section 2.1.1, an agent using the FSM decision system has a certain state at every point in time. While an FSM determines the state of an agent by going through a list of possible states the agent can reach, the HFSM uses a behavior tree or a behavior Directed Acyclic Graph (DAG) (see Figure 5). Often HFSMs use a behavior DAG, because sometimes a certain state (or behavior sub-graph) can be reached by going along different paths through the graph. In a typical HFSM scheme, the role of non-leaf behaviors is to make the decision about which of its children to trigger, while the role of the leaf behavior is to actually adopt a state. There are two general approaches in the decision making of the non-leaf behavior and at different times Halo 2 uses them both:

- The parent makes the decision based upon the conditions from the code that apply.
- The children compete with each other on relevancy or desire-to-run.

The second approach is mostly used when there are many (10-20) children, because hard coding the logic that differentiates between 20 possible children can be tedious and unscalable.

In the child competitive model Halo 2 primarily uses the prioritized-list approach to determine which child triggers. The list of children is in a priority order. The first child (with the highest priority) of which all preconditions apply, is chosen. When a sibling should win on a next update, the behavior of the child is interrupted and the behavior of the sibling is run. The prioritized-list approach uses a fixed prioritization. In order to have the ability to make priorities flexible Halo 2 uses a technique to raise the priority of lower placed behaviors temporarily whenever specified conditions are met.

Each update the decision tree runs through the conditions of the nodes in order to check whether the conditions are true in order to trigger a certain behavior. Normally, events in the game world trigger those conditions, but on certain occasions (when the condition is rarely true) it is desirable to let the event directly trigger a behavior instead of a condition. These event-driven behaviors are called Stimulus behaviors and optimize the calculation time, by skipping unnecessary checks.

Halo 2 uses different characters which can all have different behaviors. However, as most basic behaviors are equal the game uses Custom behaviors. Each character uses the same HFSM scheme, but some children can only be triggered by a specific character [17].

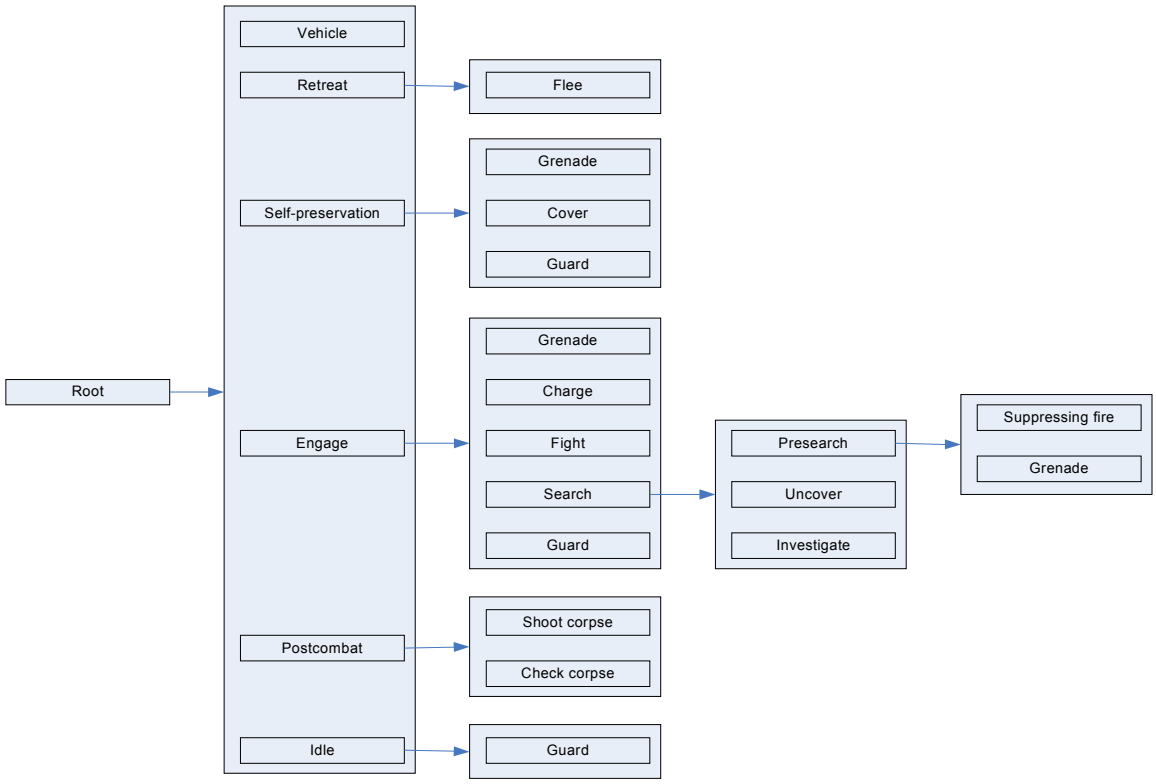

Figure 5: The behavior DAG of Halo 2.

Coordination system

Halo 2 has a joint behavior system which allows individual agents to post requests to friendly units. For instance, if an agent notices that another agent is friendly and that both agents are after the same enemy, the agent can post a request to the other agent to jointly attack the enemy. If the other agent accepts, both agents can interchange information about the enemy and the environment and work together to find the enemy and attack him together. Another example of joint behavior in Halo 2 is that an agent can ask a driver of a vehicle for a ride. If the driver of the vehicle accepts the request, he will pick up the requesting agent.

2.2 Academic work

The academic research projects described in this section either focus on the development of a decision system for single agents, or on the cooperation between agents. The final subsection discusses the use of a technique for coordinated behavior that may be called ‘observe and adapt’.

2.2.1 Soar Quakebot for Quake 2

Quake 2, released on December 6, 1997, is a first-person shooter computer game developed by Id Software. It includes a single player campaign as well as multiplayer game-modes. The network architecture of the game allowed third parties to develop AI controlled players for the multiplayer game modes [20], [31].

Decision system

In 1999, researchers at the University of Michigan, USA, developed a bot for Quake 2 which they called the “Soar Quakebot” [20]. The developers used Soar to encode the bots intelligence. Soar is a cognitive architecture to model various aspects of behavior and to implement these models. Soar is based on a production system, using rules that are roughly in the form “if A then B“ [36]. These rules are chained like a hierarchy in which abstract goals and behaviors are further decomposed & refined until they are primitive actions. For example, in order to get a certain item, the bot must go to the item, face the item, move over it and wait until it has disappeared (picked up).

The Soar Quakebot has several main tactics, which are encoded as top-level operators to start the chain of knowledge rules. The top-level operators are collect-powerups, attack, retreat, chase, ambush and hunt. In each reasoning step, Soar searches for all knowledge rules that can be applied, given its knowledge base and the partially decomposed operators. Soar does not have a built-in conflict resolution strategy. Whenever more than one rule is applicable, it creates a new sub state in which the impasse is formulated as a new problem that must be solved. On the part of the knowledge base developer, this requires explicit rules explaining the order / relative relevancy of the operators. This gives the developer the opportunity to switch behaviors based on contextual information.

2.2.2 SHOP HTN Unreal Tournament bot

Unreal Tournament, a popular first-person shooter developed by Epic Games, was released in 1999 as a follow-up title to Unreal. The game focuses mainly on multiplayer action and was launched in direct competition to Id Software's Quake 3 Arena. Unreal Tournament became an interesting research platform with the development of JavaBot, a higher-level API to the ‘GameBots for Unreal’ protocol. Using JavaBot, developers can create Unreal Tournament software players without having to worry about the specific GameBot protocol or the network interface [18], [41].

Coordination system

For his MSc thesis at the Lehigh University, USA, Hoang researched the use of planning techniques to control the strategy of Unreal Tournament bots [13]. His bots are based on the example CMU_Jbot that comes with the JavaBot distribution [18]. The CMU_Jbot includes code to move around, react to enemies, etc, but it does not have special strategies that coordinate behavior between multiple bots in games that have team objectives. Hoang interfaced an HTN-planning process (see section 3.2.1) with the JavaBot acting as a team leader, keeping track of team objectives, the locations of all bots and commanding the bots to go to certain locations.

The team game mode Hoang focused on is called domination. Simply put, teams receive points while they can keep certain predefined locations (domination points) free of enemies. The implemented strategies are based on the assumption that it is best to send each bot to a unique location. One strategy sends the bots to capture half the number of domination locations plus one, to ensure a winning position. Another strategy tries to control all points. The planning domain works for two bots only and includes separate versions for two and three domination points.

2.2.3 Observe & adapt

Another method of performing intelligent group behavior is for individuals to observe the behavior of others and to adapt their own behavior accordingly. This method is

applied to e.g. team sports where each player may be the instigator of an attack launched towards the opponent. Team mates will pick up this behavior and support it with their own actions. Intelligent group behavior is performed without explicit communications. While this approach may, for example, be of interest for developers of robots for the Robocup project[10] [34], there are a number of reasons why it is not used in computer games. In any recently developed computer game, agents all 'live' in the same simulated game world run on the same machine. Using messages to communicate plans or intentions is therefore much more practical than figuring out what other teammates are doing, which may be a very computationally intensive task. Using explicit messaging will also make the system less prone to errors since there is no way to misunderstand the intentions of fellow players.

Coordination System Category		
Centralized	Decentralized	Player Controlled
Quake 3 NOLF 2 SHOP HTN UT Bots	FEAR Halo 2 Observe & Adapt	FSW

Table 1: Coordination System Categories. The coordination system of the SOAR Quake bots (section 2.2.1) was excluded in the table, since there is no information of its coordination system.

2.3 Summary

This chapter showed some decision systems that place the HTN planner discussed in detail in section 3.2 in its developmental context. In addition, the coordination systems subsections showed the development of squad coordination systems in several games. This section summarizes the decision systems in section 2.3.1. Section 2.3.2 compares and classifies the coordination systems that were discussed in this chapter.

2.3.1 Decision systems

When analyzing the decision systems discussed in section 2.2, a part of the evolution of agents in game AI decision systems can be seen. In the beginning the AI existed of production rules. If-then rules defined all behavior of an agent. The Soar Quakebot for Quake 2 uses the production rules system to hierarchically decompose a set goal into primitive actions. Agents developed from having mere production rules to having states in an FSM. In Quake 3 an FSM was used, while Halo 2 extended the concept of FSM with a Hierarchical FSM adding several features to make it more dynamic. NOLF 2 uses an FSM as well, but states are goals that can be satisfied with a fixed set of actions. In NOLF 2 goals are not able to interact with each other. FEAR decouples the actions from the fixed goals in order to generate flexible action sets, so goals can interact. Tasks can be hierarchically assembled by the HTN planner, which is able to plan different future actions as a plan. However, the HTN planner can also be used to hard code strategies and locations as in the SHOP HTN UT bots discussed in section 2.2.2. A different approach to agent decision systems is the Observe & Adapt method used in the agents used for the Robocup.

[10] The ultimate goal of the Robocup project is to develop a team of fully autonomous humanoid robots by 2050 that can win against the human world champion team in soccer

2.3.2 Coordination systems

Coordination systems in FPS games that control the cooperation between agents can be divided into two categories: Centralized and Decentralized [38], [37], Table 1: shows the coordination systems discussed in this chapter and the categories in which they fall. Quake 3 uses a team leader to coordinate its bots. Hoangs hard coded SHOP HTN Bots are sent to certain fixed spots by a central system. NOLF 2 uses a central Blackboard System on which agents can post information. The coordination system of FEAR and Halo 2 is similar to the blackboard system used in NOLF 2. In both FEAR and Halo 2 agents invite other agents to start a joint maneuver. Agents directly communicate to other agents, which classifies the coordination systems of FEAR and Halo 2 decentralized. The agents using the coordination system in the Robocup Project observe and adapt to the new world information as they seem fit without using a central system. Full spectrum Warrior fits in neither of the categories mentioned above, as coordinated behavior between the squad members is non-existent and fully controlled by the player.

3 Guerrilla FPS AI

As described in section 1.1.2, a recent development in FPS AI is the use of planning techniques. The AI developed and used by Guerrilla has undergone similar development. The AI of the game Killzone, developed for the PS2, uses sequences of actions in combination with goal selection based on heuristics to control the behavior of NPC. For Killzone 2, the AI system was changed to make use of a specialized implementation of a Hierarchical Task Network (HTN) planner.

This chapter provides a description of the AI system of Killzone and a description of the changes that were made for Killzone 2. The first section gives a global view on Killzone's AI system, discussing everything from stimulus (observations and perception) to response (the AI decision process, behaviors and actions). For Killzone 2, the high level decision system was replaced, while the other parts remained basically the same. For this reason, the second section will focus on the changes in the decision system only. The third section described the functional abilities of the individual agent and the squad that already exist in Killzone 2 and are relevant for this project.

3.1 Killzone

When Guerrilla started making first-person shooters, the AI team developed a generic AI system that would be reusable for future games. The system was initially used for "Shellshock Nam '67". The games that followed, Killzone and Killzone Liberation, used the same system of Goals, Actions and Skills. This decision system is described in the first subsection. The subsections that follow address NPCs knowledge about enemies, the tactical evaluation of positions (position picking), path planning, and coordinated behavior.

3.1.1 Goals and Skills

The AI of Killzone is based on *sequences of actions* called *behaviors*. These behaviors can be seen as predefined plans. They are predefined in the sense that the sequence of actions is constructed during design / development phase and does not change during the execution of the game. The behaviors are plans as the actions are performed consecutively as long as none of the actions fail.

All agents have a set of *goals* that they may want to achieve while being in the game. Examples of these goals are `PursueThreat` and `AttackThreat`. At fixed intervals, the relevancy of all available goals is evaluated. Depending on world observations and data available on the character's own state, the most important goal is selected. For each of the available goals, a behavior is available that will enable the agent to reach that goal. The behavior for goal `PursueThreat` for example, consists of the actions `SelectTarget`, `PickPursuitPos` and `MoveToPursuitPos`. Like NOLF 2 (see section 2.1.2), Killzone uses character differentiation. Each character type may have its own sequence of actions to satisfy a certain goal. For example, the goal `KeepWeaponReady` may result in the behaviors `ReloadInCover` (move to cover, then reload) or `ReloadStatic` (reload at current location), depending on the character settings. Each character may only have one of these behaviors per goal.

Whenever the current action fails (e.g. a pursuit position could not be found), the chosen behavior can no longer be executed and will abort. To ensure that the same behavior will not be restarted directly, the goal that activated the current behavior will get a penalty for a short period of time. There is no explicit communication between

goals. However, during the penalty time, the goal that received the penalty will be considered less important, causing the agent to select a different goal and activate an alternative behavior.

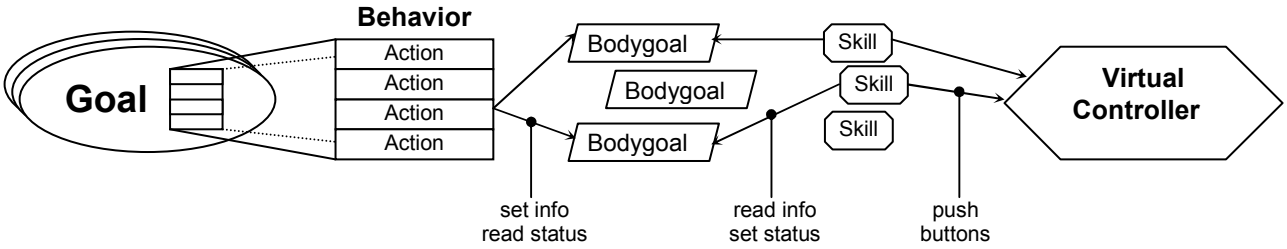

Figure 6: Overview of the process between the selection of a goal and the usage of the virtual controller. Each goal contains a behavior, a sequence of actions. Actions describe what the body should do via body goals. Skills are components that use the virtual controller according to the active body goals.

To actually get the agents to move, the AI must use a *virtual controller*. This controller contains virtual 'buttons' to do all things that the body of the agent is capable of doing, such as (among other things) to walk, run, talk, look, aim and fire. To have control over each distinct body function, agents have *skills* that, when activated, will use the virtual controller. Some of these skills use buttons that control the same part of the body and therefore cannot be combined, e.g. a body part cannot simultaneously turn left and right.

The higher level actions that comprise the behaviors cannot use the virtual controller directly. Instead they must specify *body goals* (describing what the body should do, e.g. turn the lower part of the body to face north) and assign priorities to them. A matching procedure will search for and activate a combination of skills that will satisfy the body goals that have been set by the actions. An overview of the complete process from the selection of goals to the use of the virtual controller is shown in Figure 6.

Since body functions are closely related to the type of body an agent has, various types of enemies may have different sets of body goals and skills. All humanoids share the same set of body goals and skills.

3.1.2 Combat Situation and Alert Level

Agents that walk around in the game world produce both *visual* and *auditory stimuli*. Footsteps, gunfire and explosions are examples of events that produce auditory stimuli for the AI. Among other things, various parts of the body of agents or unexploded grenades produce visual stimuli. All incoming stimuli are filtered according to the sensitivity of the receiver, the source, range and type of the stimulus. All the stimuli that one agent receives from one specific source are combined into a general indication on how well the agent perceives the source. It is through this system that agents obtain information on their opponents.

Each agent remembers information about its *combat situation*, which includes knowledge on the enemy and on their own general state of alertness or *alert level*. This information plays an important part in the evaluation of the relevancy of the various goals the agent has (see section 3.1.1).

Depending on how well an agent has perceived an enemy, it may only suspect threats (reflects belief of some enemy presence), confirm threats (reflects knowledge of some enemy presence) or identify threats (reflects knowledge on exact identity of

enemy presence). Agents store information on what they consider to be their four greatest threats. They also keep track of the threat positions and movement and will try to interpolate threat locations based on their direction and speed. Whenever one of the threats leaves sight and does not produce auditory stimuli either, agents will start to expand a range of possible locations for that threat, based on its last known waypoint and all possible directions in which the threat may have moved. The vision of the agent determines the edges of the area that is being considered. Threat prediction will stop once the threat comes into sight again, meaning that a threat could never have moved past a location that is in clear sight of the agent. If threats are not perceived for a long time, agents will lose track of them.

The *alert level* is closely related to the evaluation of threat presence. In their lowest alertness, agents do not suspect any threats. Auditory stimuli such as footsteps may not be enough to identify an enemy, but it may raise the *alert level* of the perceiver to 'threats suspected'. After threats have been confirmed, the alert level may rise even more when the agent enters combat, is under attack or is getting hit.

3.1.3 Waypoints and areas

To make path finding and position picking easier for the AI mechanism, approximately half of all FPS games, including Killzone, use *waypoints*. A waypoint is a point in the 3-D virtual world where an agent can be. If an agent or a player is not exactly at a waypoint, the closest waypoint is calculated and the agent or player counts as being on that waypoint. All waypoints in a level are connected to each other or via another waypoint by edges as in a graph. A waypoint has a certain *radius*, which is used for *visibility*. After the geometry of a level is designed, the Waypoint Exporter program is run, which among other things, calculates the visibility range in all directions of all waypoints. If a radius of another waypoint is within this visibility range, that waypoint counts as being visible from the waypoint calculating its visibility range.

Killzone uses *AIAreas* to group certain waypoints in the entire grid of waypoints. An AIArea may contain one or more waypoints that are usually connected to each other. AIAreas are most commonly used to name certain locations, e.g. a certain room. Agents can have *area restrictions* that restrict them from entering or leaving certain predefined AIAreas. Area restrictions are often used to gain more control over the autonomous behavior of the agent, e.g. to keep an agent within a certain area or to prevent the agent to move through an area and plan its path around the area.

3.1.4 Position picking and path finding

While squad orders and scripted positions can make an agent move to a waypoint as well, the autonomous AI agent chooses its positions based upon threat information. The agent defines a primary threat and secondary threats. As threats are moving, the agent has to continuously pick the best position to deal with the threats. When in combat, short distance movement is best, because while moving the agent is more vulnerable for attacks. The best option is usually a covered position from which the agent can attack its threats. These conditions can be calculated by an evaluation function using the following criteria:

- Proximity to the current position of the agent.
- Positions having line of fire to the primary threat.
- Positions having cover from secondary threats.
- The fighting range is the distance range from which the agent prefers to attack.

In Figure 7 the agent has evaluated all surrounding waypoints within a certain radius. The agent picks the next position to move to according to the highest evaluated value [39].

Agents in Killzone use the *A* algorithm* [24], [12], to find a path between two waypoints in order to move to the picked position. This path always exists, because the position picking algorithm has already picked legal waypoints. However, as scripted positions have not been calculated by the position picker, they can be illegal, i.e. not be reached, causing the agent to stall.

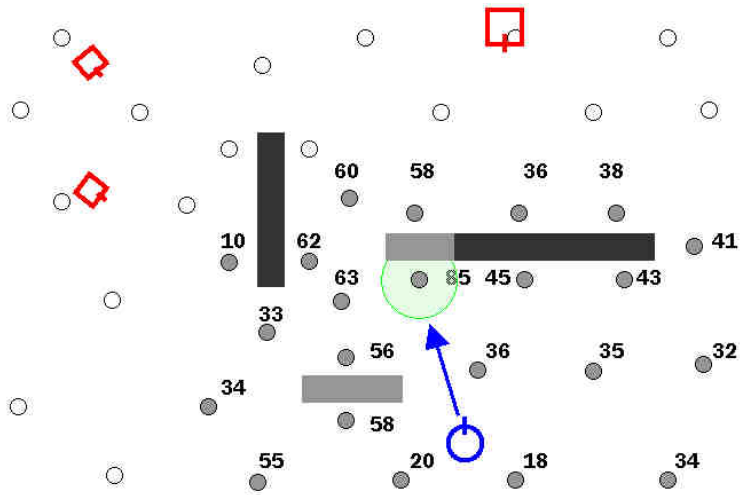

Figure 7: The agent (blue) picks the best position according to the highest value of the evaluation function. The big red square is the primary threat. The small red squares are the secondary threats. The dark rectangles are high walls and the lighter rectangles are low walls. [28]

3.1.5 Order priority

In addition to the autonomous behavior of the individual agents, individuals can also receive orders from the squad (see section 3.1.6) or the script (see section 1.1.3) to perform. To cope with all the different behaviors the individual should perform at a specific point in time when having its own desired behavior and having orders to perform, individuals have a strict ordering in the behaviors the individuals can perform. Squads can receive orders from the script as well. These orders have higher priority than the autonomous squad behavior. Orders can be given the following priorities:

- Blind following, always execute the order without regarding the environment.
- Follow orders, only postpone the execution of the order in case of direct danger, e.g. when in the danger area of a grenade or when there are enemies at very close range.
- Non-battle initiative, only execute the order when no enemies are in the neighborhood or there is no other plan to execute.

The following example illustrates how an individual would act with a combination of internal priorities and orders. The squad orders a group of individuals within the squad to line up in front of a door with ‘blind following’ priority. The individuals start to move to the door, since the squad order overrides the individual autonomous behavior. Halfway their movement, one of the individuals receives a different order with ‘follow orders’ priority to move to a specific spot in the corner of the room the rest of the individuals is doing their line up in. While individual that received the new order starts executing it, the rest of the individuals keep moving towards the line up positions initially given by the squad. While all individuals are moving towards their spots, an enemy throws a grenade into the area all individuals are moving in. All members are in the danger area of the grenade. The individuals that were given the squad order which has the ‘blind following’ priority ignore the grenade and keep moving towards the spots designated by the squad order. However, the individual that received the order with ‘follow order’

priority postpones the order and immediately lets the autonomous behavior of the individual move the individual out of the danger area of the grenade. After the grenade has exploded, that individual resumes its order with 'follow orders' priority, moving to the spot designated by the order.

3.1.6 Squads

To allow for coordinated behavior, the AI of Killzone supports the grouping of individual agents into squads. Squad in Killzone can be seen as an extra layer of control that is placed on top of individual agents. The relation between a squad and its members is hierarchical. Each squad may have any number of members (typically not more than four), while each individual agent may only be a member of a single squad.

The squad establishes a binding that acts like a hub that enables sharing of information between squad members. It also provides a combat situation that acts as a summary of the combat situation information from all members. All members of a squad share their information on enemies. As soon as one of the squad members identifies a threat, it will inform its squad. The squad will save this information in its own combat situation and will pass it on to all other squad members.

Even though the squad does not have a visual representation of its own, it is an agent as well. Squads have their own set of goals and behaviors that allows them to execute a number of coordinated actions using their members. Most of these goals are only executed when activated through a game script. As with individual agents, commands can be sent to squads and these commands may be bundled in a command queue. The squad will try to execute the orders one by one. Other than giving commands, scripts may also manipulate the area restrictions of a squad, restricting all members of the squad simultaneously.

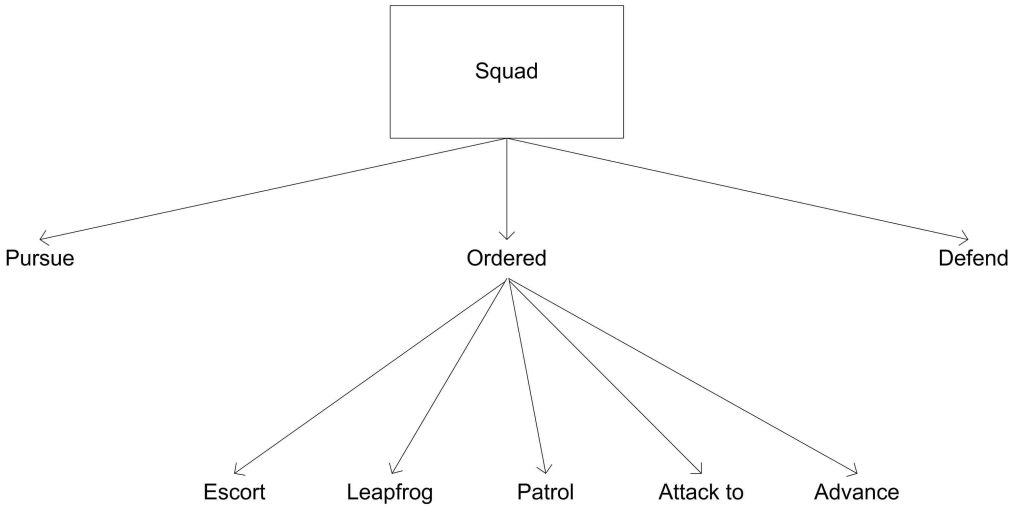

Figure 8: The squad behavior in Killzone and Killzone 2

When creating a squad, a formation must be specified. During all movement and during defense operations, the squad will try to maintain this formation. Each squad uses its own area of operation to keep its members together. During movement, the center waypoint is used for path planning. The squad will then project its formation onto the planned path resulting in new positions for all squad members. Once all squad members arrive, the squad moves its area of operation a little further and starts projecting the formation once more. This cycle repeats until the squad reaches its movement target.

Figure 8 shows Killzone's squad behaviors. `Pursue` allows the squad to pursue enemies, by moving towards last known locations of threats. `Defend` is the default behavior and will cause the squad to defend its current location, while trying to maintain its formation. `Advance` is the squad equivalent of the ordered move command. `Attack_to` is basically the same as `Advance`, with the exception of the squad members stopping and fighting once they encounter enemies. `Patrol` is a looping version of `Attack_to`. `Escort` is meant for escorting the player. During this operation, the squad is invisibly tied to the position of the player by means of what could best be described as a rubber leash. The last ordered behavior, `Leapfrog`, will cause half of the squad to advance while the other half holds position, provides covering fire. Once the first half of the squad reaches a certain distance, the roles are reversed.

3.1.7 Bot AI for Domination games in Killzone

Killzone includes AI modules to control the bots for all available multiplayer game modes, including Domination. Since the AI programmed for Domination will serve as a standard to test and evaluate the deliverables, this section will explain the inner workings of this AI system.

The coordination between the bots for Domination is centralized. Each individual bot of the same faction receives orders from and reports to the same *faction leader*. Like squads in Killzone, this faction leader is a reasoning component without visual representation in the game world. The faction leader is implemented as a script that is allowed to update each game update cycle. The faction leader assigns a *role* to every bot. At any time, each bot is either an attacker or a defender of a specified target. These targets are also handed out by the faction leader.

The number of bots that is assigned a role as attacker depends on the number of captured switches. Whenever the faction leader is updated, it decides how many attackers are desired based on the following rules:

- When all switches are captured: 0% attackers.
- When at least one switch is captured: between 50% and 75% attackers. This is determined linearly. The more targets are captured, the fewer attackers are desired.
- When no switches are captured: 100% attackers.

The faction leader does not keep track of dead bots; each bot that spawns will get a new assignment. Bots spawn at a randomly assigned spawn point and are assigned an attacker role by default. Like the spawn locations, the targets are also chosen at random, with the exception that neutral targets are preferred over switches which have been captured by the enemy.

When a bot is assigned the role of attacker, the faction leader orders it to move to a location in the vicinity of the target switch and then to report back to the faction leader. When attackers arrive at the switch they send a message to the faction leader. Subsequently the faction leader either assigns the bot the role of defender, if the switch was successfully captured, or it assigns the bot the role of attacker of a different, randomly chosen target.

During the faction leader's update, the excess of defenders is changed into attackers to get to the desired number of attackers. This is done by repeatedly finding the best defended switch and assigning an attacker role to one of its defenders.

Except for sending out move commands, the faction leader does not control the behavior of the bots. The actual attacking behavior is defined on the individual level using Goals and Actions (see section 3.1.1). Defending bots use the same behavior,

but their movement is restricted by the faction leader using area restrictions (see section 3.1.3).

The review of the source code of the Domination bot AI of Killzone has brought to light that there are some programming errors that could cause the bots to behave differently from what was originally designed. These errors cause the faction leader never to assign defender roles, not even after successfully capturing a switch. This leads to much more chaotic behavior than intended. The problems are addressed in section 6.2.1 in preparation of the experiments that compare the Domination strategy with the new bot AI.

3.2 Killzone 2

The AI system of Killzone 2 is based on the system of Killzone. Many of the lower level AI functions remained the same. The most important change (and the most relevant change to this project) was the replacement of the goal-behavior system (as described in section 3.1) by a task planning mechanism. This change enabled a greater range of possible NPC behaviors and improved the flexibility of behavior implementations. The following subsection will explain the planning approach in general, followed by a section on how the HTN planner replaces the goal / action structure of Killzone's AI system. The third subsection provides details on how the planner acquires and requests information about the game world. The fourth and last subsection discusses the implications of applying a planning mechanism to a dynamically changing environment.

3.2.1 Planning using Hierarchical Task Networks

Hierarchical Task Networks (HTN) [4] is an approach to planning in which tasks are planned in the same order in which they are later executed. The result of the planning process, the plan, is a sequence of concrete actions, called *primitive tasks*, which need no further elaboration to be executed. To construct this plan, the planner takes a planning problem and a planning domain. A planning problem is formulated as a *compound task* that needs to be decomposed (planned) combined with a set of facts that describe the current world state. Planning domains contain knowledge about the hierarchical relations between compound and primitive tasks as well as the order in which subtasks must be executed. The structure used to formalize this knowledge is called a *method*. An example is shown in Figure 9. The syntax of the Killzone 2 HTN domains is based on that of the SHOP HTN planner [26], which adopted the syntax of the language it was implemented in, i.e. LISP[11] [3].

The example method shown in Figure 9 specifies how the compound task 'eat' with a single argument '?food' may be decomposed. Each *branch* of the method specifies a valid decomposition. (Eating food may either be done by using a fork or by using a spoon.) The decomposition may be a single primitive task, such as in this example or any other combination of primitive and / or compound tasks. For each compound task that is used within the decomposition part of a branch, there must be another method that specifies how that task can be decomposed. All branches also specify a (possibly empty) set of preconditions. During the planning process these preconditions are matched against the world state. The matching process is similar to that of Prolog [2], instantiating unbound variables during the process. This way, information can be

[11] Lisp is a family of computer programming languages with a long history (first appeared in 1958) and a distinctive fully-parenthesized syntax.

retrieved from the world. In this example, both branches specify a single precondition, testing for the presence of a fork and a spoon, retrieving their value in the process.

```
(:method (eat ?food)                    < Method head
  branch1                               < Branch name
  (have-fork ?fork)                     < Precondition
  ((!eat-with-fork ?food ?fork))        < Decomposition

  branch2                               < Branch name
  (have-spoon ?spoon)                   < Precondition
  ((!eat-with-spoon ?food ?spoon))      < Decomposition
)
```

Figure 9: Example HTN method

The planner will always select the highest / first branch for which the preconditions can be met. If we have both a fork and a spoon, the planner will create a plan that uses the fork. As a result of this rule, there is no need for other conflict resolution mechanisms.

The planning process is recursive. Compound tasks are recursively decomposed until the planner finds a sequence for which all tasks are primitive. When a branch cannot be expanded to the level of primitive tasks, the planner will start backtracking and try the next branch of that method.

In JSHOP, a reimplementation of SHOP in Java [14], the definition of a method may include a specification of the effects it would have on the world state. So called add and delete lists specify which facts should be added or deleted from the world state. For example, the method shown in Figure 9 may have the precondition `(have-food ?food)` and the same fact in its delete list. In other words, performing the task `(eat ?food)` will cause the `(have-food ?food)` fact to be deleted. The exact outcome of the execution of a task in Killzone 2 is unpredictable, therefore its HTN planner does not support add and delete lists.

The HTN planner of Killzone 2 supports a controllable way of manipulating the world state. Its HTN planner supports specialized primitive tasks for adding and deleting world facts. The primitive task `(!remember ?period ?fact)` puts `?fact` into the HTN database for `?period` amount of seconds or forever, which is denoted by a dash. `(!forget ?fact)` deletes that specific `?fact` from the HTN database.

3.2.2 HTN planning in Killzone 2

In Killzone, agents evaluated the relevance of goals and adopted behaviors that were coupled to the goals. In Killzone 2 the HTN planner is used for these two steps. The knowledge domains specify how tasks are performed, but also which (sub) tasks should be considered more important than others. Each type of agent uses a combination of generic and specialized knowledge rules. Generic knowledge rules for e.g. basic movement are shared between agents. Defining characteristics of the behavior of each character type are specified in specialized parts of the domains.

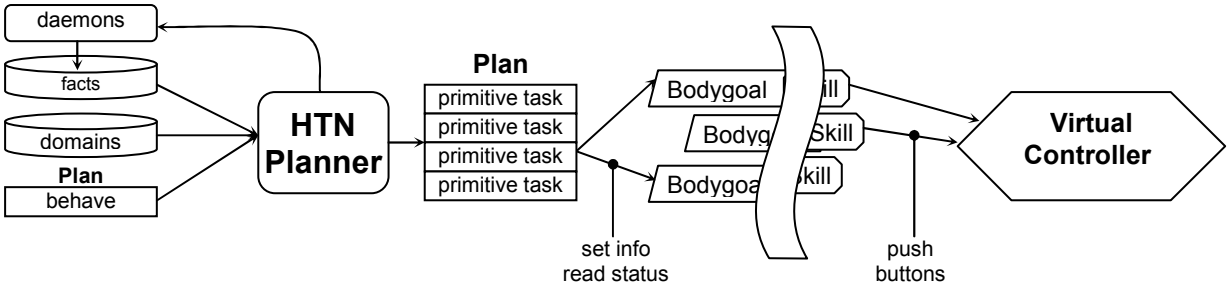

Figure 10: Overview of the construction of a plan consisting of primitive tasks. Daemons provide world information as facts. The domains consist of rules that define allowed decompositions for each compound task. Initially, each plan contains one compound task: behave. Using the facts and the rules from the domains, the HTN planner decomposes this initial plan into a sequence of primitive tasks. Primitive tasks describe what the body should do via body goals. The process from body goals to the use of the virtual controller remains unchanged with respect to Killzone (see Figure 6).

Independent of the type of character, all plans start out the same; as a single compound task (behave), which means that the agent should do something useful given the current situation in game. Higher level choices in the HTN domain decide *what* the AI should do. These choices can be compared to the heuristic evaluation of goal importance in Killzone. Subsequently, lower level choices decompose the selected compound task(s) into a sequence of primitive tasks, deciding *how* the agent should perform the task it selected. In this setting, the role of primitive tasks is roughly equivalent to that of actions in Killzone. Examples of primitive tasks are '!fire_burst_at_threat' and '!walk_segment'. Each character may use any number of primitive tasks as long as they are compatible with its body type. The plans that result from the planning process can be seen as more dynamic versions of the behaviors of Killzone. They are more dynamic since, depending on the context within the game world, the decomposition of a compound task may result in several different plans.

Apart from the primitive tasks, each character also uses a set of *daemons*. These daemons are processes that monitor relevant world state information for one agent. Just before the planner is started to construct a new plan for an agent, all daemons are requested to update their status and translate their findings into data structures (database facts) that can be used by the planner. More about daemons can be found in section 3.2.3 below. An overview of the process from the construction of a plan to the use of body goals is shown in Figure 10. The process from body goals to the use of the virtual controller remains unchanged with respect to Killzone (see Figure 6)

The HTN planner is not only used for the control of individual agents, it is also used for the coordination of squads. As mentioned in section 3.1.6, each squad has an invisible squad leader that uses the planner to coordinate squad actions and / or movement. Squad plans mainly consist of series of orders that are sent to the squad members. At the time of the start of this project, there were no squad domains for the HTN planner. As such, the squads did not exhibit any behavior of their own, neither ordered, nor autonomous. Squad members only shared information on their threats as described in section 3.1.6.

3.2.3 HTN Planner communication with the C++ code

A *daemon* in Killzone 2 is C++ code that runs once every agent update (see section 3.2.4). Daemons are able to read and manipulate the HTN database. Daemons 'translate' the world situation to information that is usable by the agent in terms of HTN facts. An example of a daemon is `DaemonObjectStates`. Every update

`DaemonObjectStates` checks the states of objects. For instance, the state of a door can be either `open` or `closed`. `DaemonObjectStates` translates the states of the object, in this case the C++ door object, into a fact that can be read by HTN planner usually in the form `(object_type object_name object_state)`, e.g. `(door door_1 open)`.

A *request* is a specified fact put into the HTN database by the HTN planner. That specific fact can be read by the daemon, which can act accordingly. An example of a request is the fact `request_path`. When `request_path` is `true`, a daemon can act by planning a new path for that agent. Requests have to wait one update cycle to complete, because the HTN planner has to wait for the daemon to be run, as daemons always run at the start of a new cycle. A more direct way of communicating from the HTN Planner to the C++ code is a *call term*. A call term is executed immediately. For instance, the call term `(call add ?a ?b)` can add two numbers (in this case `?a` and `?b`) and return the result.

3.2.4 Applying HTN planning to a dynamically changing environment

In an FPS like Killzone 2 planning ahead is difficult because of the changing environment. Threats are constantly changing positions and objects can change states, e.g. a door opens or closes, changing path planning options. Long plans tend to invalidate quickly, so agents in Killzone 2 make local plans for a short term. The agents replan at a rate of five times per second (one update) to keep the plan up to date. Squads replan at a rate of two times per second (one squad update). The time between two generated plans is often too short to execute a whole plan. There are two causes for the current plan to be rejected; the plan can become invalid due to a changing environment or a better plan makes the current plan less desirable. As replanning happens often, the agents need strategies to replan efficiently. Different strategies to optimize replanning include *updating of tasks*, *subtask priority order changes* (see section 2.1.4) and *partial* or *delayed decomposition of tasks*.

When a replanning agent is performing a task, which is still viable in the next update, it would cost much calculation time replanning for the same task. A strategy to tackle this problem is when an agent is replanning and comes across the same task he is already performing, to simply update the task to match the present situation. For example, when an agent is performing the task to move to a cover position, while the enemy presence changes in such way that another cover position provides better cover from the threats than the chosen position, the task to get into cover remains the same, but is updated with the new cover position.

The idea behind applying partial decomposition in combination with delayed decomposition of tasks is to use calculation time efficiently. Partial decomposition only decomposes the part of a plan that is relevant at this moment, delaying the decomposition of the rest of the plan until that part becomes relevant. To test whether the plan is still relevant after executing the first part of the plan, the outcome of the first part is matched against the preconditions of the rest of the plan. If the rest of the plan never becomes relevant, that part is discarded and minimal calculation time is wasted, because that part of the plan was never decomposed [42]. The partial and delayed decomposition optimization strategies were never implemented in Killzone 2.

4 Technical design

4.1 Design overview

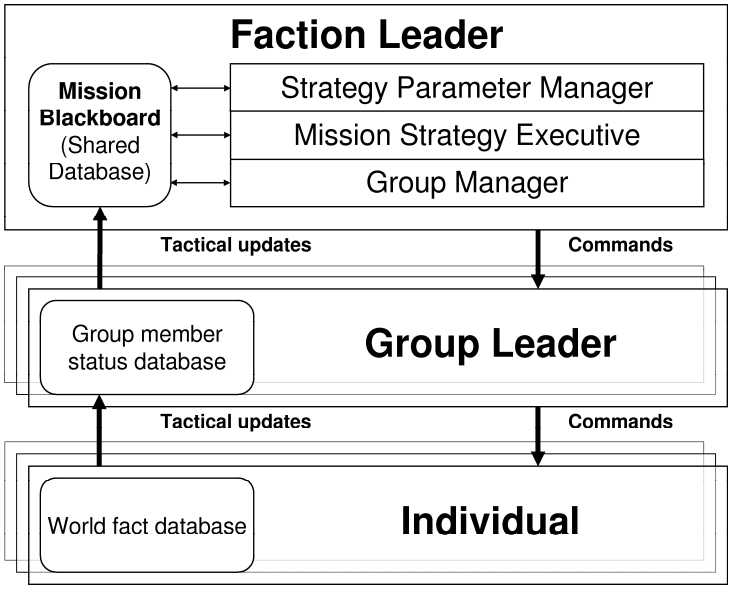

Figure 11: System structure overview.
Group and squad leader are used as synonyms.

The system that controls the behavior of the bots can be split up into three levels. Figure 11 shows how the three layers of the system are linked together by several communication channels. The three levels form a hierarchical structure similar to the one found in military organizations. Specific behavior of individuals is mostly due to the orders they receive though the chain of command. As in military context, the chain of command is the line of authority and responsibility along which orders are passed. Higher ranked components have more responsibility than the lower ones and orders that come from a superior must always be followed. Orders start at the higher ranked AI components and are made more specific as they are delegated downwards. Like in the army, the chain of command also implies that the highest and lowest layers do not communicate with each other directly.

For each faction there is one mission general taking care of the strategic decisions for all AI players of that faction. Its task is –among other things– to decide which of the mission objectives should targeted. Directly under these faction leaders are squad leaders. They are responsible for a smaller number of soldiers. However, they do need to give more specific orders to their subordinates to make organized offensive or defensive actions possible. The lowest layer comprises only individual soldiers. Since they do not have subordinates, their task is to execute the squad leader's plans for attacking and defending objectives.

4.2 Faction leader design

Faction leaders have several distinct tasks. Their main task is to select which targets among all available mission objectives should be attacked or defended. Different mission types will have different kinds of objectives. Some kinds of objectives may need to be destroyed or killed, which means no defense force is needed once the action succeeds. For Capture and Hold, the main task for the faction leader is to

choose how many and which of the Capture and Hold areas should be targeted either for offense or defense. To support this decision, this component collects and records tactical data about the game in progress. Only data that is also available directly or indirectly to all human players may be used for this purpose. When evaluating this criterion we may assume that all human players of one faction may actively communicate using voice chat and reason about their collective status. Current scores of both factions and the location of the home base spawn areas are available to anyone on screen. In addition, the faction leader may use mission dependant information. For Capture and Hold this comprises the exact quantity and location of all Capture and Hold areas. Human players that are acquainted with the level that is being played, will know these locations as well from experience. With this information, faction leaders may use the distance to the nearest spawn area as a criterion for selecting targets.

Besides target selection, this component is also in charge of finding a way to divide all available bots over a number of squads in such a way that there is one squad for each task that needs to be executed. If necessary, it will create new squads and put unassigned individuals in these squads. However, without consideration of the current division of bots over squads, many of them may be unnecessarily reassigned to different squads in unneeded exchanges of soldiers from two or more different squads.

To overcome this issue, every time the faction leader decides to do something new, squads that are currently already in operation (perhaps already performing one of the required tasks) are kept untouched as much as possible. When the number of squads and the division of bots over these squads is satisfactory, the faction leader's task is to send out its orders to attack, defend, patrol, scout, etc. For Capture and Hold it orders squads to move to Capture and Hold areas.

The last task that this component has to perform is the selection of spawn points for respawning NPCs. For each of the squads, the faction leader selects a spawn point that fits their task. Generally, this is a spawn point nearby the target of that squad, but it does not need to be. Squads will spawn all of their members at this spawn area.

4.3 Squad leader design

The following subsections describe the requirements for the squad leaders that control the bots. The first subsection will explain which types of squad behavior should be supported and which ones are not included. These behaviors are further elaborated in the second sub section, which defines squad tasks. Subsection three discusses the information required by the squad leaders to perform their tasks, while the last section describes how this information is communicated to and from the squad leaders.

4.3.1 Types of squad behavior

There are a number of characteristics by which coordinated squad behavior may be categorized. As a first characteristic, the behavior may be a form of synchronous action, where all of the squad members perform the same action at the same time. This is the simplest form of coordinated behavior since the squad leader may simply broadcast a specific order to all members. Bots that all start charging an objective at the same time display an example of this behavior. On the other side, different bots in one squad may be given distinct roles in a joint behavior. In this case, the squad leader will have to send out different orders to the various members. An example of this behavior is a flanking operation where part of the squad attacks directly from the front, while others walk around and attack from the side.

Apart from executing orders – which is a form of reactive behavior, squads may also be programmed to display autonomous more pro-active behavior. In this case, the squad leader is the initiator of squad actions. This project will not be exploring these types of behavior, since, by allowing squad leaders to initiate squad movements or behaviors, the control over the bots will become more difficult. More about research on autonomous squad behavior can be read in [1].

A last category of coordinated squad behavior comprises actions that are normally already part of the behavior of individuals, but are now coordinated. Most of these cases concern coordination of the choice of targets. As an example, squad members may coordinate their attack targets so they can concentrate their fire on the most pressing threats. When there are no threats, squad members may divide the locations at which enemies are likely to appear, so that each squad member watches a different location, making sure that the squad will pick up new threats, regardless of where these threats are coming from.

4.3.2 Squad tasks

The squads need to be able to perform any task that they can receive from the faction leader and any subtask that may be part of those tasks. From a complete system's point of view this means that the squads should take care of coordinating actions such as moving from A to B, defending objectives, attacking objectives, following targets and various combat moves. Some of these actions can be very general, suitable for reuse for multiple mission modes (combat techniques and movement for example) while other actions may be very mission specific. When looking at the Capture and Hold mission specifically, squads need at least to be able to attack a Capture and Hold area, to capture it and to defend it from being captured by the enemy. To ensure that the members of a squad can attack as a group, squad members will need to regroup at a nearby location that provides cover from the neighboring areas. If squad members regroup before starting their assault, they will arrive at their attack location mostly simultaneously. This will increase their chances of success.

The squad leader should have a better overview of the level than the individual bots. When relevant information is at hand, squad leaders should provide the members with any information that would prevent the squad members from unexpectedly entering enemy territory.

4.3.3 Required information

To perform its tasks, squad leaders will need multiple sources of information. On the mission independent side, they need to have access to the status of squad members, because during an attack, squad tactics may change. If the number of alive squad members drops beneath a certain threshold, the squad leader may decide to abort the attack. In other cases, the squad leader may decide to wait for reinforcements. To assure this kind of control, squad leaders should be able to reason about all squad members. It should know of all members if they are either alive and playing, or dead and waiting to spawn.

Furthermore, for Capture and Hold, they have access to the exact location of all Capture and Hold areas, what their sizes are and what their current status is, i.e. which faction is the current owner and for neutral areas if it is being attacked. Squad leaders also know of the areas around Capture and Hold areas that should not be crossed when attacking another area. Allowing the AI to reason with this kind of information may be considered cheating. However, more experienced players that already played

in the level multiple times will remember these locations and the position of the objectives as well.

4.3.4 Squad communication

Apart from receiving and sending orders, there are two more communication channels that squad leaders need to handle. Individual bots can send tactical updates on their perception of the current situation. The squad leader's task is to process these messages and send a summarized version of this information back to the faction leader. It should also inform the faction leader of any major changes in the progress of executing the ordered task (i.e. the task was completed or cannot be completed any longer).

4.4 Individual design

Since most of the individual behavior of the bots can be done by a system that has already been designed for single player AI, there are less issues left in the design of the individual AI then there are for the upper two layers. The behavior of the individual AI bots does require some adjustments however, to make them act more aggressively, lowering their restrictions and raising their health to equal human players (see section 1.2.2) As with the squad leader, the AI must support the execution of all tasks that may be received, this time from the side of the squad leader. Any mission dependant atomic actions – handling mission dependant objects, destroying objects in ways other than by firing – should be designed and implemented on this level. For Capture and Hold, individuals need to know the exact locations of the capture areas and be able to approach and capture the areas. Individuals need a way to pick locations within the target capture area. To spread out a bit more, bots should move to different spots in the area they are attacking. If the attack succeeds the bots should not all be standing at the same spot, since this would harm an efficient defense.

Since the AI lives inside the game's simulated world, it uses a virtual joystick instead of a physical one (see section 3). Instead of using an analogue stick, the AI can specify an aim target directly. In order to make the AI miss convincingly (and not look stupid) there are several systems in place. Aim 'mistakes' are simulated as well as the spread of a weapon. Preferably, bots should aim just as well as average skilled human players would. Making their aim worse would make them look stupid, while making the aim better would create a feeling of unfairness in human players.

Individual bots must be able to communicate messages to the squad leader concerning tactical information, about their own status and about the task they are performing. Since the squad leaders will already notice when one of the squad members dies, squad members do not need to send a failure message when death coincides with the failure of the task. Individuals do need to send updates on tasks that succeed or fail without measurable consequences for the squad leader. Tactical information can be anything concerning enemy troops, for example information on perceived enemies at a certain location or on enemies that are killed while defending an objective.

5 Implementation

The structure of the three layers is implemented as follows. The faction leader is programmed as a C++ class, while the squad leader implementation uses HTN domains. The strength of the HTN planner lies in the evaluation of logical conditions and in the recursive decomposition of compound tasks. Although iterations and more complex arithmetic operations can be performed, HTN solutions to these kind of problems are considered suboptimal to those of a procedural language such as C++, since they require more programming effort and are generally less readable. The faction leader needs to evaluate heuristics and needs to perform algorithms to assign individuals to groups. Since these problems are best solved using procedural arithmetic, the HTN planner is considered less suitable or at least less convenient for the implementation of the faction leader. Aside from an iteration over its squad members, squad leaders have little need for iterations or arithmetic. To explore possible advantages of the planner the squad leader's AI has been implemented as an HTN-domain. These advantages are further discussed in the evaluation section 7.1.2.

The three layers all operate on different update frequencies. The individual's HTN planner is invoked about five times a second, while squads are updated only twice in that same period. The faction leader is only updated once every 6⅔ seconds (that is once every 100 game logic cycles). The updates of two faction leaders are interleaved. This means that when the ISA faction leader gets updated, the Helghast faction leader is updated 3⅓ seconds later, and then 3⅓ seconds after that, the ISA faction leader is updated again, etc. (See Figure 12) As higher-level plans do not invalidate as quickly as the plans of individual bots, the faction leader can suffice with updates performed at this frequency. In fact, a lower frequency prevents some problems with faction leaders constantly switching between two or more strategies when the conditions for selecting a strategy change rapidly.

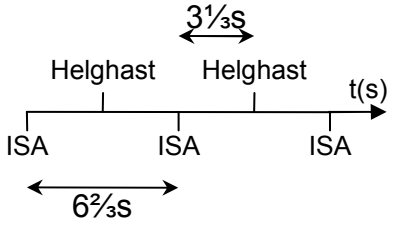

Figure 12: The faction leader updates are interleaved.

The three layers communicate in a hierarchical fashion. Any two layers may only exchange messages if one of them is the direct subordinate of the other. Between the three layers, there are four communication channels to implement.

The faction leader sends orders to the squad leaders. These orders indicate which of the objectives should be targeted by the squad. The orders are inserted in the squad's command queue and remain active until either cleared by the squad (when the order has been completed) or until they are overwritten by new orders from the faction leader.

Squad leaders send orders to individual squad members. These orders are inserted into the individual's command queue and, just like the squad orders, remain active until either cleared by the individual (on completion or failure) or until they are overwritten by new orders from the squad leader.

Individual soldiers inform their squad leader whenever they completed an order that was issued by the squad leader. Individual soldiers also share information on threats they encounter. A message is sent whenever the individual

- encounters an enemy for the first time.
- loses track of an enemy.
- observes one of his threats has been killed.

The messages sent by the individuals are not queued, though they are saved until the next squad update. The squad leader uses the feedback from the individuals to update their record on the squad members' states and to update its combat situation. (see section 3.1.2)

Squad leaders inform the faction leader whenever the squad members engage in combat and they inform the faction leader when an order that was sent by the faction leader cannot be completed. Both messages are delivered using a callback mechanism on the faction leader. The messages are processed directly and influence the behavior of the faction leader the subsequent updates.

The following subsections will elaborate on the implementation of the strategy layer, the squad layer and the individual layer.

5.1 Faction leader implementation

The faction leader's logic is implemented in a C++ class. This allows for direct access to game data needed in the decision making process. Part of the logic is mission independent and is therefore implemented in a base class that may be shared over multiple types of faction leaders.

Figure 13: Excerpt of Figure 11, showing the main components of the faction leader.

Figure 13 schematically shows the base components of the faction leader. By design, the three main components of the faction leader do not need to be executed in sequence, or even at the same frequency. For example, if the calculation costs for the *Strategy Parameter Manager* are high, the update frequency of this component may be lowered as compensation. The other two components will just use the last calculated parameters. For the implementation of Capture and Hold however, the three components all depend on the evaluation of tactical information on the Capture and Hold areas. The base components of the faction leader are implemented as three methods, which are called in sequence every time the faction leader is allowed to update, but only after the tactical information on Capture and Hold areas is evaluated.

The first part to execute, the 'Strategy Parameter Manager' determines the global aggressiveness of the strategy. For game modes that have a single target for each of the factions, the Strategy Parameter Manager would calculate the number of desired attackers versus defenders For capture and hold it calculates the desired number of captured targets (capture and hold areas). This is the amount of areas that are attacked and / or defended simultaneously. The *Mission Strategy Executive* selects

targets and creates tasks and a desired division of forces over these tasks in terms of soldiers per task. The output of this component is used by the third part, the *Group Manager*. This part creates squads, reorganizes the bots to fit in these squads and sends out the tasks that need to be executed.

In addition to this sequence, the faction leader contains a number of methods that are called on a request (callback) basis. This is because the timing of these methods cannot be synchronized with the update frequency of the faction leader. The game spawns new players at set intervals. Since the faction leader is only updated once every 6⅔ seconds, it cannot determine spawn locations for bots in advance, as events in the running game may change the available spawn areas within those 6⅔ seconds, invalidating the previous spawn area choices of the faction leader. To circumvent this problem, the logic that assigns bots to their spawn area is called on request at the moment players are allowed to spawn. Other callbacks are used for processing messages sent by squads. By introducing a message queue, this last part could become synchronized with the faction leader update. However, for the current implementation there was no benefit in combining the processing of multiple messages.

For each objective, the faction leader keeps a record of tactical information. This comprises information used in target selection and spawning. The information provides answers to the following questions, indicating which objectives have the best chance of being captured successfully, or which objectives should otherwise be preferred:

1. *Is the area currently captured?*
2. *Is the area neutral?*
3. *Is the area currently targeted (for attack/defense) by a friendly squad?*
4. *What is the nearest usable spawn location to the area? At what distance?*
5. *How many times did a squad get attacked while targeting the area?*

The answers to the top two of this list can be determined directly. The answers to number 3 up to 5 are updated by several parts of the faction leader: An area is considered targeted once the order has been given to capture it. This indication remains until either the area is successfully captured, or the squad has indicated that the task to capture the area has failed. The nearest spawn area is reevaluated every update by the ‘Strategy parameter manager’. Whether or not a Capture and Hold area is targeted depends in part on the distance to this nearest spawn area. Number 5 on this list is only influenced by the messages received from squad leaders.

Most of the information the faction leader uses to make its decisions is highly dynamic and thus has to be queried every time. The locations of spawn areas and objectives however, do not change during the execution of the game. The distances between spawn areas and objectives are most important to the faction leader’s logic. It will search for the closest spawn area every time a spawn area has to be picked for a selected target. Instead of constantly recalculating those distances and sorting the results, the distance calculations and list sorting are done at initialization. Subsequently, for every Capture and Hold area, a list is constructed of all spawn areas augmented with their distance to that specific Capture and Hold area. The resulting table can be used to quickly find the closest spawn area for an objective.

The following sections describe the inner workings of the most important parts of the faction leader's implementation.

5.1.1 Strategy Parameter Manager

The Strategy Parameter Manager implementation for Capture and Hold does two things. It determines the number of captured areas that are needed to prevent the other faction from winning and it evaluates tactical information for every Capture and Hold area.

A simple heuristic for the number of areas needed to win the game is half the number of areas plus one. The score gained by capturing and holding this number of objectives will ensure victory. Unfortunately this only holds under the proposition that areas are not recaptured and that the score balance will only be influenced by the captured areas. This is rarely the case. The paragraphs below describe how the desired number of areas is calculated, taking into account the current difference in score and the rate at which the enemy earns points for kills.

When a Capture and Hold area has been captured, the team that has ownership receives points every *tick*. *GainPerArea* in Figure 14 represents the amount of points that is gained by capturing an area and holding it until the end of the game round.

$$GainPerArea = \frac{TimeRemaining}{TickLength} \times PointsPerTick$$

Figure 14: The number of points that can be obtained by capturing an area and holding it until the end of the game.

For Capture and Hold, captured areas are not the only source of points. The Strategy Parameter Manager must take into account the number of kills made by both teams and must compensate the desired number of areas accordingly. Initially, the bots would increase and lower the desired amount of areas based on how well the bots performed at making kills. However, in one of the test games, the bots lowered their desired number of areas to one, thinking that they had already won the game. This caused them all to gather around a single area which in turn caused an unforeseen drop in the points that were collected for kills. The bots managed to play even, but it showed that the prediction needed to be adjusted.

The prediction assumes that the score awarded for making kills is increasing at a constant rate throughout the game for both teams. The final formula only uses an *EnemyGainByKills*, which represents the potential lead which the enemy will get by acquiring points for kills (if any). If the bots acquire more points on kills than the enemy, this is not taken into account.

The calculation of the *DesiredNumberOfAreas* assumes that areas are always captured by either team. Effectively, areas which are not captured by ones own team, are assumed to yield points for the enemy. Assuming neither team has a lead, the required number of areas to prevent the enemy from winning the game round is half the total number of areas. If either of the teams has a lead, this may affect the number of required areas. Figure 15 shows how the EnemyGainByKills is predicted and how the DesiredNumberOfAreas is determined.

$$EnemyGainByKills = \mathrm{Max}\left(0, \frac{EnemyScoreByKills - OwnScoreByKills}{TimeElapsed} \times TimeRemaining\right)$$

$$DesiredNumberOfAreas = \mathrm{Ceil}\left(\frac{NrOfAreas}{2} + \frac{EnemyScore + EnemyGainByKills - OwnScore}{2 \times GainPerArea}\right)$$

Figure 15: Heuristics to determine the number of Capture and Hold areas that must be captured and held until the end of the game to prevent losing the game round.

There is another task this part performs. It updates the tactical information on the objectives. It uses the distance table and the data on which areas are captured by which faction to determine the best available spawn area for each of the objectives. Together with the chosen spawn area it also stores the distance. The spawn area selected by this process is used to spawn the bots. By updating the tactical information, areas that have been successfully captured are deregistered as attack targets.

5.1.2 Mission Strategy Executive

The Mission Strategy Executive takes care of target selection, task creation and a division of forces to carry out these tasks. It controls how many bots go where to do what. It takes the desired number of areas from the Strategy Parameter Manager, subtracts the current number of captured areas and selects that many targets.

Objectives that are currently already targeted are considered most important. This prevents swapping of targets after the orders to capture an area are sent. Next are neutral targets. At the start, all objectives are neutral, meaning that neither of the factions has captured the area. During the game, it may happen that a contested objective is left unguarded by both factions (after a fight), making it neutral. In both situations capturing this area will probably yield a source of mission points without much effort. Attacking an area which is captured by the enemy will both yield a source of mission points and take away that same source of points from the enemy, but at greater costs. Because of the smaller required effort and the assumption that neutral areas would be captured by the enemy otherwise, neutral areas are preferred targets.

Another criterion used to select targets is the number of attacks that squads have encountered while targeting this area. This includes the situation where an area is already captured and a squad is defending it and the situation where a squad is sent out to attack an area. This mechanism is intended for whenever attacks launched at a certain objective fail repeatedly. In this case, another target will be chosen. To prevent twitching by this mechanism, it will only influence comparison if the difference in registered attacks is more than 10. The last criterion, but one of the most important ones is the distance to the best / nearest spawn area. Objectives that have a smaller distance are preferred.

When the targets have been selected, the number of bots available is divided over the selected targets. The default strategy implemented for Capture and Hold, hereafter designated as *Group strategy*, will assign scouts to those areas that were not selected as targets. These scouts are sent to ensure that no enemy objective is left unguarded. Subsequently the other bots will be divided evenly over the targets. To prevent remainder bots (i.e. 7 bots / 2 targets) from constantly being reassigned, areas of equal interest are always considered in the same order. For example, for 8 bots, 3 objectives,

2 of which are targets this will result in a division of 4-3-1. Figure 16 shows this task division as the active one in a bot test game.

Figure 16: In game debug view of a bot test game in progress. The interface shows the current division of bots over the Capture and Hold areas. (CAHArea1: 4 bots, CAHArea2: 3 bots, CAHArea3: 1 bot)

5.1.3 Group Manager

The Group Manager takes the tasks and the division of forces as produced by the Mission Strategy Executive and applies it to the current state of the game. It does so by creating or removing squads, reorganizing their members and order squads to perform their tasks.

As with the selection of targets, this part is implemented with the goal of continuity of the AI's behavior in mind. It will keep all squads that are already doing one of the tasks requested by the Mission Strategy Executive. This combines nicely with the Mission Strategy Executive's system of keeping the same targets active while they are still actively being attacked. Together these rules will make sure that no squad ordered to attack an area will be sent new orders until they either succeed or signal their defeat.

When making sure that there is one squad for each task that needs to be executed, squads that already exist are preferred over the creation of new squads. Members of existing squads are likely to be near each other. By keeping them on the same teams they will not have to cross the level in order to join another squad. This could otherwise decrease their chances of survival. After this step, any abundant squads are disbanded.

When the number of tasks and the number of squads is equal, the sizes of the squads are checked against the desired sizes indicated by the Mission Strategy Executive. Members in squads that are too large are removed from their squad. To prevent the removal of members that are in the middle of executing orders sent to them by their squad leader (causing the appearance of indecisive behavior), dead squad members are preferred, followed by squad members that are not executing orders. The bots that are freed are subsequently reassigned to the squads that are too small. An

example of input and output of the three main components of the faction leader can be found in Figure 17.

Squads that need new orders are informed of this fact before they actually receive the new order. The warning (an order to 'start a new operation') and the order itself are sent in a sequence, queuing them at the squad leaders side. Processing the order to 'start a new operation' will cause the squad to reset its own status (see section 5.2.2) in preparation of a new task. Subsequently, the squad leader will be given its new task. Since the multiplayer squads do not behave proactively, the priority of the orders is set to the highest level.

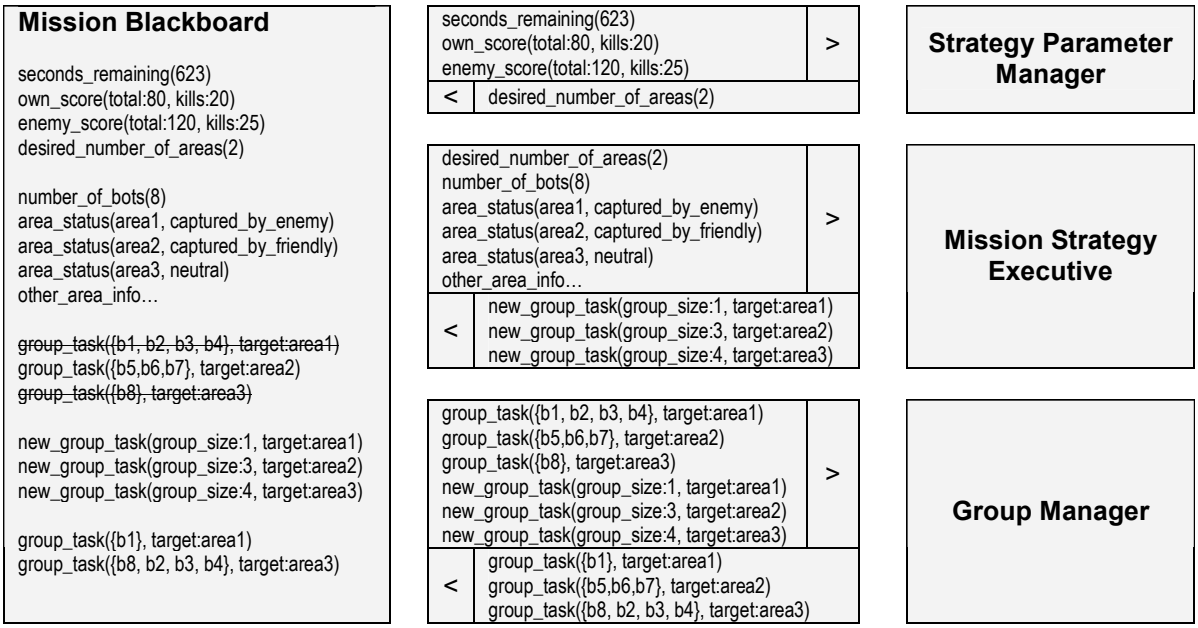

Figure 17: Slightly simplified example (some area information has been left out) of the input / output of the three main components of the faction leader for Capture and Hold. The Strategy Parameter Manager determines the desired number of targets based on scores, the Mission Strategy Executive selects the targets and generates a task division. The Group Manager takes the current task division and reassigns bots to other groups as needed, (partially) overwriting the old task division. The Group Manager also orders the group leaders to capture the area assigned to them. Individual bots are denoted as b1-b8.

5.1.4 Callbacks

The faction leader's implementation has two callbacks – subroutines that are not executed as part of its regular update routine. These callbacks exist to enable the faction leader to respond to or to reason about certain events directly as they appear. One callback method selects spawn points for dead bots. The other callback processes messages sent by squad leaders.

Spawn point selector

The spawn point selector is called at the moment players are allowed to spawn. Its task is to assign spawn points to each of the bots that must be respawned. Between the last faction leader update and the spawning of the bots, some of the Capture and Hold areas may have been captured by the enemy and as such are not usable as spawn point anymore (see section 1.2.4). This is why the tactical info on each of the objectives is updated before any selection takes place. The spawn point that is assigned to dead bot depends on the current task its squad is trying to execute, as well as on the progress of the squad in executing that task.

As a rule of thumb, bots are best spawned at the spawn point that is closest to the target of their squad. In this context, the distance is used as an estimate of the risk of getting killed while trying to reach the target. Bots that have to move over greater distances generally have a higher chance of getting killed before even reaching the objective.

The nearest available spawn point can be directly retrieved from the tactical information on the objective that that bot's squad has targeted. (see section 5.1). For Capture and Hold, it may not always be the best option to spawn at the nearest available spawn point right away. When the nearest available spawn area is far away; bots could take quite a long time to reach their objective again. Bot tests that are not further discussed in this paper have shown that by the time the bot reaches the battle area again, the outcome of the fight is often already decided. The squad may have failed in capturing the objective in which case squad members need to regroup before trying again, or the squad may have successfully captured the objective in which case it needs to set up a strong defense to hold the objective. Both squad regrouping and area defending are explained in section 5.2.2.

Given that regrouping is a less urgent operation than establishing a defense for an objective that has just been captured, it is best to postpone the spawning of bots until the outcome of the fight has been decided. This delay is implemented in the spawn point selector. While the attack on the target objective is still in progress, dead members of the attacking squad do not receive a new spawn location, which effectively prevents them to spawn too early.

Squad message receptor

The second callback processes any messages sent by squad leaders. For Capture and Hold, the squad leaders may send two kinds of messages:

- `(engaged_in_combat)`
  This message indicates that the squad encountered new enemies and has engaged in combat. When this message is received, the faction leader will take note of the squad being attacked by incrementing the appropriate counter for the objective targeted by the sender. This will give a negative bonus to the target objective which will affect subsequent target selections.
- `(task_failed)`
  This message is sent by squad leaders to indicate that the last task assignment from the faction leader could not be completed successfully. For Capture and Hold, this means that the attempt to capture the objective failed. When this message is received, the objective in question will be un-targeted, allowing the faction leader to switch targets the next update.

5.2 Squad leader implementation

As described in 5.1.3 Group Manager, the faction leader will send the selected target as an order for the squad leader. The squad leader targets the area that was last received from the faction leader. For Capture and Hold, the action of attacking an area will, in case of success, always be followed by the defense of the captured area. The faction leader does not send an explicit order to capture or defend an area after it has been attacked and all enemies have been defeated, these orders are all implicitly given by the initial order to target a certain area. It is because of this that the squad leader never clears the order to target a certain Capture and Hold area, unless it has been given orders to target another one.

To switch targets, the faction leader sends out a new sequence of orders, overwriting the current one, starting with an order to 'start a new operation'. This first order will cause the squad leader to discard any data related to the operations centered on the previous target area. Subsequently, the second order will reinitialize the squad leader in order to start advancing towards the new target area. This solution is specialized in handling and targeting of permanent objectives like Capture and Hold areas. Objectives that must be killed or destroyed will require a different protocol. Unlike the Capture and Hold task, the order to kill or destroy such objectives can be completed since those tasks may end once they have been successfully executed.

After a target has been passed on, it is the squad leader's responsibility to coordinate the movements and combat behavior of the squad members in order to attack, capture and/or defend the target area. It must tell the squad members where and when to regroup and when to attack. It does so by issuing orders to the squad members. Orders to move to specific locations, orders to attack Capture and Hold areas, orders to report back to the squad leader and orders to stay within a certain set of area's – avoiding danger areas – while doing all this.

Since for the implemented strategies, the squad only uses actions that require squad members to follow orders blindly without question, orders sent by the squad leader always have the highest priority. From the rules of the game (see section 1.2.4) mode Capture and Hold, it follows that capturing and/or defending areas is more important than making individual kills or for individual bots, to stay alive.

During its operation the squad leader also uses feedback acquired from squad members. For example, in order to keep track of the status of the squad member, instead of constantly querying the status, the squad leader orders individuals to return a message once another task has been performed. Squad members that have just been added to a squad, inform their squad leader of this event. Additionally, squad leaders receive messages from members when enemies are being spotted, are lost or killed.

The last kind of communication that squad leaders need to handle is squad messages. These are messages sent to the faction leader to aid it in making strategic decisions. Whenever the squad leader decides that it can no longer execute the task ordered by the faction leader, it will inform its superior. It will also send a message once every time the squad engages in combat.

The logic of the squad leader is mostly implemented as an HTN domain. Every update, squad leaders update their own state and traverse all their members to see if they need (new) orders or if their recorded state needs to be updated. The structure of the HTN domain is explained in section 5.2.2. The HTN knowledge-rules use information and specialized routines provided by several daemons. These daemons are specialized in providing squad member- and multiplayer information to squad leaders. The squad daemons are explained in section 5.2.1. Furthermore, by grouping bots in a squad, several individual actions can be coordinated without explicit planning. These behaviors are explained in section 5.2.3.

5.2.1 Squad Daemons

The multiplayer squad leader uses several daemons. Some of the information provided by these daemons is inserted to the squad leader's database every update. Other information is too time expensive to compute every time and is therefore provided only on request.

The most general squad daemon can be used for squads in single player games as well. It converts squad member messages and inserts them into the database. Both fixed messages about enemies and custom messages are converted to database atoms.

Another daemon is meant specifically for multiplayer squad leaders. Multiplayer squad leaders differ from squad leaders used in single player games in that they are commanding AI players instead of one-life AI soldiers. Whenever a bot is killed, it rejoins its squad once it has spawned again. Every squad uses a daemon that provides a list of (living) individuals that are currently a member of the squad. To allow multiplayer squad leaders to reason about all their members, the multiplayer squad daemon provides a list of all AI players that are a member of the squad, also specifying if these AI players are currently alive and playing or dead and waiting to respawn. Additionally, this daemon shows which level is currently being played (to allow for level specific tactics) and which mission or game mode is currently active (to allow for squads that support multiple mission types).

Besides providing generic multiplayer information, this daemon also inserts mission specific information into the squad leader's database. Depending on the mission that is currently active, this daemon provides information on the mission objectives. For Capture and Hold it provides data on all Capture and Hold areas, listing their name and state. This state indicates whether or not the area is captured or being captured and which of the factions are involved. Instead of referencing the actual factions, the daemon presents the information in terms of friend and foe.

An area may have any of the following states: captured by friend / foe, being attacked by friend / foe, being captured by friend / foe or otherwise unknown. The status of an area only comprises information that human players may also derive from the feedback they receive from the HUD. Whenever a team of human players would not be able to derive an area's state, the daemon will list its state as unknown. This may happen for example in the case of a neutral area being captured by the enemy.

The information that is only available on request concerns the regrouping of squad members. To start the regrouping action, squad leaders request a regrouping waypoint which is selected from a list of candidate waypoints based on the current positions of all group members. Subsequently it requests an ETA of the bot that is furthest away from the chosen waypoint. This value is used as a timeout for the regrouping behavior.

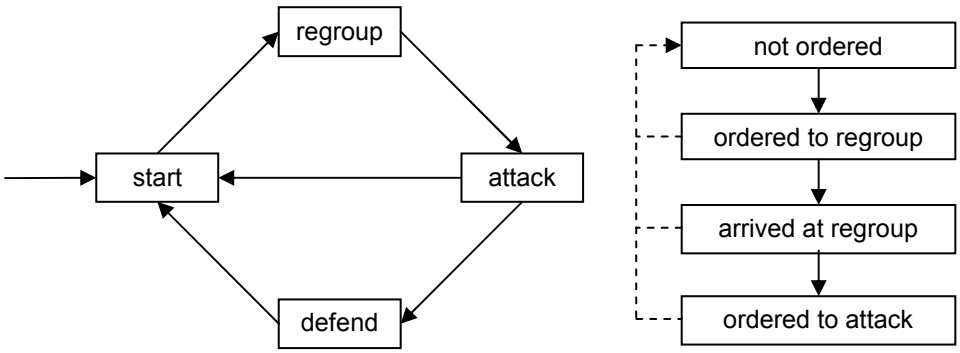

Figure 18: Left: Capture and Hold squad states
Right: Capture and Hold squad member states

5.2.2 Squad HTN Domain

The multiplayer squad HTN domain used for Capture and Hold contains level specific information on regroup locations and on which areas are considered dangerous depending on which objectives are captured by the enemy. When a squad is created, the squad leader's HTN domain initialization rules will fire, remembering this level specific information in the database. All subsequent updates will cause the planner to construct a plan that both updates the internal state of the squad leader and possibly sends orders to every squad member.

The squad leader keeps track of its own state as well as the state of each of the squad members (concerning the execution of orders). The rules in the domain specify the transitions between these states. The left diagram in Figure 18 provides an overview of the squad leader FSM, displaying all states and transitions. The right diagram displays the states of squad members as recorded in the squad leader's database. Whenever the squad leader receives the order to start a new operation it will reset itself to the starting state, regardless of its current state. Additionally, all recorded states of squad members will be reset.

Once a substantial part of the group (more than half of the squad) is accounted for, the regrouping will start. Members are ordered to a specific regroup location and their state is updated accordingly. Members that arrive at their location message the squad leader, which causes the member state to be updated. Once all members have arrived, or once the action times out, whichever comes first, the state is changed to attack. In this state, all members still waiting for orders are ordered to go to the Capture and Hold area and once again, their state is updated. Squads that only comprise one bot skip the regroup phase since there is little purpose in regrouping one soldier.

Since there are two possible outcomes for a fight, there are two state transitions from this state. Either more than half of the squad gets killed and the squad is considered defeated (not depending on the number of enemies), or the squad manages to start capturing the target area. In the first case, the squad will reset to the starting state. Remaining bots that are still attacking continue in such a case, as they would most likely not survive a retreat. This means that the state of these squad members is not cleared. However, newly spawned bots will first go though the regroup process again before attacking once more. The faction leader is informed of this event, which allows the faction leader to assign the squad to a new target. In the second case (the attack was a success), the state changes to defend. Since the change of faction of a Capture and Hold area is directly visible to the faction leader, squad leaders do not need to inform the faction leader of this event. Changing to the defend status has no direct implication on the current state of squad members as they do not need new orders. As long as the area remains an active target for the squad, the squad members will stay in the area. The defensive behavior of the individuals consists solely of the default infantry attack / defend behavior. Improvements on this behavior are discussed in section.

Changes in squad member states and squad leader states usually coincide. For example, when changing from regroup to attack, all members are ordered to the target area. There are some additional rules that reset squad member states for members that died, for members that just respawned and for bots that were removed from the squad altogether. This may happen when bots are reorganized to other squads.

Besides the orders to move to specific locations and orders to return messages the squad leader also sends out orders to stay within certain areas within the level. To prevent bots taking a path through an occupied Capture and Hold area that is not their target, squad members are ordered to stay away from any non-target Capture and

Hold area and its surroundings. To look up the right areas, the squad leader uses the member's current areas and the current target combined with the level specific area information supplied at initialization.

5.2.3 Non-planned coordination

Besides the coordinated behavior resulting from explicit planning and ordering by the squad leader, squad members can coordinate some of their individual actions in a more simple way. This type of coordination is based on direct sharing of data between squad members, for example the distances to common enemies.

Target selection greatly depends on the distance to the enemy. Enemies that are close are often more important than enemies that are further away since they can hit with greater accuracy. Normally each squad member chooses its target solely on its own perceptions and distances to the enemies. By incorporating the distances of the enemy to other squad members as well, multiple squad members will conclude that the same enemy soldier is their greatest threat and should be targeted. By concentrating their fire this way, the enemy forces will decline more rapidly.

A similar solution is used for dividing scanning waypoints over multiple squad members. It ensures that squad members scan different locations. The validity of both constructions is based on the assumption that human players within the same team may freely exchange information about their location and their enemies.

5.3 Individual implementation

For the most part, the multiplayer bots are controlled by the same knowledge base as the one used for the behavior of AI soldiers in the single player game. In contrast to single player AI soldiers, which are primarily meant to be smart looking cannon fodder, multiplayer bots should act more like simulated human players. (See 1.2.2 Bot AI versus single player AI) To accomplish this, bots are set never to display relaxed behavior and to always suspect enemies. To reflect human movement, the bots run at maximum speed as long as they did not identify any threats. Bots use weapon settings identical to human players (weapons shoot just as good/bad as for human players) and receive an amount of health points identical to human players. The bots' perception and aim tries to approach that of an average player.

Additional code enables the individual bots to execute the orders sent by their squad leader; capturing an area and returning messages to their squad leader. A special daemon provides a call term that will select a position within a requested Capture and Hold area. This daemon randomly selects waypoints from the set of available locations inside the area. A specialized task enables the explicit sending of squad member messages.

6 Evaluation

In this chapter, the multiplayer bot system and the Group strategy for the game mode Capture and Hold are evaluated. Section 6.1 discusses a series of test rounds that were performed in which teams of human players battled against the bots. The feedback from these tests provide information on the entertainment value and difficulty of the bots. Section 6.2 discusses a comparison between the Group strategy and the Domination strategy of Killzone. Section 6.3 shows by example, how much code may be reused for the implementation of an AI strategy for another game mode.

6.1 Test rounds

Two of the evaluation criteria as specified in section 1.6 can only be tested by means of game tryouts. By organizing several test rounds with different teams and processing their feedback, we acquire an indication of the entertainment value and the difficulty of the bots. In addition, direct feedback from test players can tell us if the players take notice of the organized behavior of the bots. Section 6.1.1 describes how the tests were set up and performed. Section 6.1.2 explains which questions were asked to gather feedback. Section 6.1.3 discusses the results of the test rounds.

6.1.1 Test rounds setup

At the time of the development of the bots, no one outside Guerrilla or Sony was allowed to see Killzone 2. Therefore, employees of Guerrilla were used as test subjects. The bots were tested against three human teams, namely members of the game design team, the art team and the game code team respectively. The first two teams played three consecutive rounds and the game code team played two rounds. (Due to technical problems, not related to the AI and outside the scope of this thesis, their third round was cancelled.)

All test rounds had a team of 8 human players, all playing as ISA, battling against a team of 8 bots, all Helghast. For the purpose of making screenshots and video captures of the game in progress, a ninth Helghast player was used as an observer. All test games were played on Southern Hills, the only mission being a 15 minute Capture and Hold.

Since the bots did not have any behavior for using grenades, these were excluded from the game. The only available weapon was the standard rifle. To make sure neither of the teams would get the advantage of starting early, the game was not started until all players were ready. At the end of each round, the final score was logged. At the end of the final round, the team of participants sent their feedback.

6.1.2 Feedback questions

To get the best out of the feedback, each of the participants was asked to answer a number of questions. Instead of asking directly if the bots were fun or hard to play against, we have asked them to answer the following questions:

- What is your general impression on the bots?
- What kind of strategy or tactics did you notice?
- Did their behavior look organized? (please comment)
- Did their behavior look intelligent? (please comment)
- Did their behavior look human like? (please comment)
- What did you think of the difficulty level of the bots?

- Do you have any suggestions?

By asking these questions, we attempt to indirectly find an answer to the question whether or not it is fun to play against the bots, without 'steering' the answers of the participants. Bots that display recognizable behavior, intelligent behavior or organized behavior are fun (more) to play against. If we find that the participants recognized organized behavior, or perhaps even classify the behavior of the bots as intelligent, this is an indication that the bots are fun to play against.

In any game of competitive nature, opponents that are challenging but not invincible are the most fun opponents. Finding out how hard it is to beat the bots is therefore a part of determining their fun factor. Most employees of Guerrilla exceed the skill of a novice player of FPS games. At the date of these tests, all these employees had been attending multiplayer play tests (without bots) weekly for about three months. This means that if the contestants think the bots are hard to beat, novice players will have an even harder time beating them. On the other hand, if the contestants think the bots are easily beaten, this does not necessarily mean that novice players would think the same.

As a last question, asking for suggestions may reveal the contestants' opinions on which part of the bots' behavior could be improved.

6.1.3 Results test rounds

Out of the 24 participants, 18 responded and sent answers to the questions. Not all of the respondents answered all questions separately, and some of them provided additional remarks in their answers that were not directly related to the questions. Therefore, this section does not discuss the feedback by reviewing the answers per question, but instead focuses on difficulty, entertainment value and otherwise the most common remarks in the participants' feedback. Quotes are marked anonymously with the respondent's number and department name.

Round	Human team	Bot team	Ratio
Design 1	345	133	39%
Design 2	355	108	30%
Design 3	338	104	31%
Art 1*	97	89	92%
Art 2	275	132	48%
Art 3	192	192	100%
Code 1	280	129	46%
Code 2	269	137	51%

Table 2: The final scores of the test rounds. The ratio shown is the bot team score divided by the human team score. The ratio is provided to give a quick overview of the bot team scores compared to the human team scores.
*) The final score for this round was not recorded. This is the last known score at 8:06 / 15:00.

Test rounds scores

The final scores of all test rounds can be found in Table 2. The bots did not win any of the test rounds, but did manage to play even with the art team once. The three teams performed quite differently. The best human team seemed to be the design team. This may have something to do with the fact that they were located in the same room and

communicating, giving each other orders while playing the game. The difficulty as reported by each participant, however, did not seem to be related to the score ratios of the participant's teams. Most of the respondents (13) characterized the difficulty as *"challenging"*, *"about right"*, *"tough but beatable"*, *"pretty solid"* or *"medium"*. A minority (4) reported that they found the bots *"pretty hard"* or *"hard to beat"*.

"I think [the difficulty] was about right, I felt they were tough but beatable. So you don't get the feeling that it's unfair and you still get some sense of achievement for killing them." (design/3)

Awareness of cooperative behavior

All but one of the respondents noticed the bots' grouping behavior. They saw the bots moving around and attacking in groups. Some noticed the bots defending Capture and Hold areas as a group. A smaller number of respondents also reported that they noticed that the bots divide their forces amongst multiple targets. One respondent noticed that the bots wait to regroup before initiating an attack.

"They seemed to wait till they weren't alone before moving. And some remained to guard the bases." (design/3)

"They tended to group very well, it seemed rare or never that a lone AI would be charging in somewhere. We also learned that if we fended off an assault on a particular base that we could expect one of the others to be under attack shortly (if not already)." (design/4)

Human like / Intelligent

Most respondents thought the grouping of the bots made their behavior look human like and / or intelligent. The fact that they often ignored threats and did not use cover when traveling made the bots seem much less intelligent and human like according to eight of the respondents. This is most likely caused by the fact that the squads use orders at the highest priority (as described in section 5.2), since capturing or neutralizing the objective is more important than killing the enemy. This does however make the bots less human like. The target selection described in section 5.2.3 which was introduced to increase the squads' chances of survival also seems to have some unexpected non human like results. The way the squads move was said to be *"ant like"*, as all squad members choose identical paths and end up walking directly behind each other in a long line. Apart from that, the most mentioned remark is that the bots did not seem to have any notion of chokepoints or entry locations for the areas they were guarding.

"Sometimes they would seem to charge an objective and not deal with near by threats. Some times they would ignore me if I was not firing and someone else was even though I was a lot closer." (design/1)

"They are acting like ants sometimes, coming in large lines." (art/13)

"To defend a spot they would just stand in one spot and scan. (…) Humans would defend the chokepoints to an objective area before the actual objective itself." (design/4)

Accuracy

More than half of the respondents (11) commented on the shooting accuracy of the bots. However, apparently, the opinions on the accuracy of the bots are quite diverse. One respondent from the design team characterized the aiming of the bots as *"very life like"*, while two others from that same team commented *"They were obviously bots. Often very accurate when I wouldn't be."* and *"They were remarkably accurate."*. In the art team, two respondents approved the aiming skills of the, while four others again classified the bots' aiming as *"superior"* or *"unnaturally good"*.

In the code team, respondents mentioned *"They aren't accurate enough at long range vs. (almost) static targets"* and *"On one-on-one shoot outs they lost. Also I noticed that they were very good in aiming at long distances, but only in open areas"*.

It seems that the optimal aiming skill for the bots is dependant on personal preference and skill of the human player. It may be worth investigating how the optimal aiming skill of AI controlled players can be determined, or how a difficulty setting could be used to accommodate both novice and advanced players.

Entertainment value

As part of the development of the multiplayer game modes of Killzone 2, multiplayer builds are tested regularly. In many of these test games people just run around without actually playing the game mode. During the multiplayer bot tests however, participants started to play in a much more organized and strategic fashion. Since the bots played in such an organized fashion, this forced players to cooperate more themselves. The players of the design team, which were mostly in the same room at the time of the bot test, actually communicated by shouting orders and information on the bots behavior to each other. Besides these observations, ten of the eighteen respondents explicitly mentioned that they enjoyed testing the bots, calling them *"[very/good] fun"*.

"They are pretty solid! It was good fun to play against them. Luckily we managed a draw the last game…" (art/10)

6.2 Strategy comparison

This section describes how Killzone's AI for the multiplayer game mode Domination was translated and tested against the new Group strategy for Capture and Hold. The results of these test rounds should answer the question if the changes and enhancements in cooperative behavior provide the new strategy with an advantage. Section 6.2.1 discusses the translation of Domination to the new system, section 6.2.2 describes the test setup and 6.2.3 visualizes and discusses the results.

6.2.1 Implementing the Killzone Domination strategy for Killzone 2

In order to make a comparison between the Domination strategy (as described in section 3.1.7) and the new Group strategy possible, the Domination strategy had to be 'translated' in order to fit into the new bot architecture. This section describes how the Domination strategy fits into the new system. It also addresses the problem that caused bots never to defend switches. and it introduces some changes to the Domination strategy that makes it up to date with the new rules of Capture and Hold.

Translating Killzone's Domination into a Strategy for Killzone 2

For Domination in Killzone, all individual bots were controlled by one central process. This process translates without many problems to the faction leader structure described in section 4.2. The flow of the Domination strategy is based on a desired number of attackers. The function of this variable is much like that of the desired number of captured areas for the Group strategy. Like the desired number of areas, the desired number of attackers is calculated in the Strategy Parameter Manager.

In contrast to the new strategies, the Domination strategy does not use groups / squads. Therefore the faction leader's implementation does not include a Group Manger. There is also virtually no reason to include the layer of Group Leaders, except for the fact that that individual bots cannot directly communicate with the faction leader. A single 'squad' is used to facilitate in the communication. It functions as an

intermediary, passing through all messages from individuals to the faction leader. The faction leader on the other hand can send orders directly to individual bots.

Under normal circumstances for the new strategies, the faction leader creates squads when needed and assigns orders to these squads immediately. The squads remember their orders and commence the execution as soon as squad members have spawned. For Domination however, this does not work, as it does not use squads. As long as the individual bots are not spawned, they cannot be ordered. To cope with this problem, the faction leader uses a list of bot *players* instead of spawned bots to keep track of assigned roles and targets. (This is similar to the squad keeping track of dead squad members.) During its update, the faction leader selects a target and assigns the attacker role to dead bots. Whenever a message from the squad confirms that the bot has been spawned, the faction leader will send out the orders according to the previously selected target.

Since we would like to attribute any improvement in performance of the new strategy to the new coordination of the bots, we must rule out differences between the two tested strategies with respect to individual behavior. No attempt has been made to mimic the properties of the individual behavior of Killzone's bots playing Domination. On the level of individual behavior, the tested strategies are using identical knowledge domains.

Repairing the Domination strategy

The logic of the original Domination faction leader would only allow an attacking bot to change role if there was another defending bot that could change its role to attacker. However, it would change defenders into attackers to get the balance between attackers and defenders right, even before considering changing attackers to defenders. As a result, bots would never defend switches. By reversing the order, the faction leader now changes attackers into defenders at captured switches before restoring the balance of attackers versus defenders by removing defenders from the best defended switches. Since capturing an area takes more time than capturing a switch in Killzone, the logic was changed slightly to make sure that Domination bots can actually capture an area. They no longer switch targets immediately when they arrive at an area which has not (yet) changed faction.

Game mode Domination versus Capture and Hold

An important difference between the Domination game mode and Capture and Hold is that the latter allows bots to spawn at captured areas as well as at spawn points that reside within their home base. This increases the value of captured areas as reinforcements may spawn at these locations and, as a result, these areas they may be used as a secondary base from which an assault can be launched. The new Group strategy incorporates rules to take advantage of this feature. To prevent an unfair advantage of the new strategy with respect to the use of spawn areas, the strategy based on Domination uses spawn areas the same way. Bots are spawned at the area which is nearest to their target.

6.2.2 Performing strategy comparison

To test if the new strategy outperforms the domination strategy from Killzone, the two strategies were set to battle against each other, without intervention from human players. The game was set to automatically restart after each round. At the end of each round (once the time limit had been reached), the score of each faction was logged.

Figure 19: The spawn selection overview of the Southern Hills multiplayer level. Each dot indicates the position of a player. ISA are indicated in green, Helghast in red. In this setup, the Helghast split up into two groups + 1 scout, while the ISA are using a strategy that makes them operate in one large group. (see section 6.3.1)

The test rounds were all performed on the Southern Hills multiplayer map, playing a 15 minute game of Capture and Hold. Even though the layout of this map is symmetrical, there are small differences between the ISA and the Helghast side of the level. As these differences could cause, for example, the ISA to arrive at their designated target just a few seconds before the Helghast, these differences could interfere with the test results. To even out this bias, half of the tests were done with the ISA playing the domination strategy and the Helghast playing the new Group strategy, while the other half of the tests had the strategies switched.

6.2.3 Results strategy comparison

To help interpret the data and to draw some conclusions, the data needs to be presented in a way that makes the difference between the two strategies apparent. Since there are many ways in which the data could be presented, we will discuss a few of them and elaborate on our choice.

When visualizing scores, it may be intuitive to choose for the score ratio. The ratio can summarize the results by showing by which percentage the new strategy outperforms the old one. Unfortunately, the ratio cannot be used in a histogram to give a more detailed overview of all test rounds. This is caused by the asymmetric view that the ratio provides. In a histogram using bins of equal size, a ratio of 1.0 – 1.2 does not mean the same thing as a ratio of 0.8 – 1.0 does for the enemy. This problem can be solved by adapting the bin sizes, however, non uniform bin sizes would make the overview less readable.

Contrary to the ration, the score difference is a symmetrical measure of performance of both. A score difference of 20 would mean the same thing as -20 would for the enemy. The score difference can be used in a histogram, but it lacks the quality of the ratio in indicating if a game was won / lost in a close match (200 to 220) or if the game was won / lost by far (20 to 40).

To get the best of both measurements; the symmetry of the score difference and the ‘achievement indication’ of the ratio, the performance is defined as follows:

$$StrategyPerformance_{AB} = \frac{Score_A - Score_B}{Score_A + Score_B}$$

Figure 20: Symmetric strategy performance measure. Positive values indicate strategy A outperformed strategy B and vice versa.

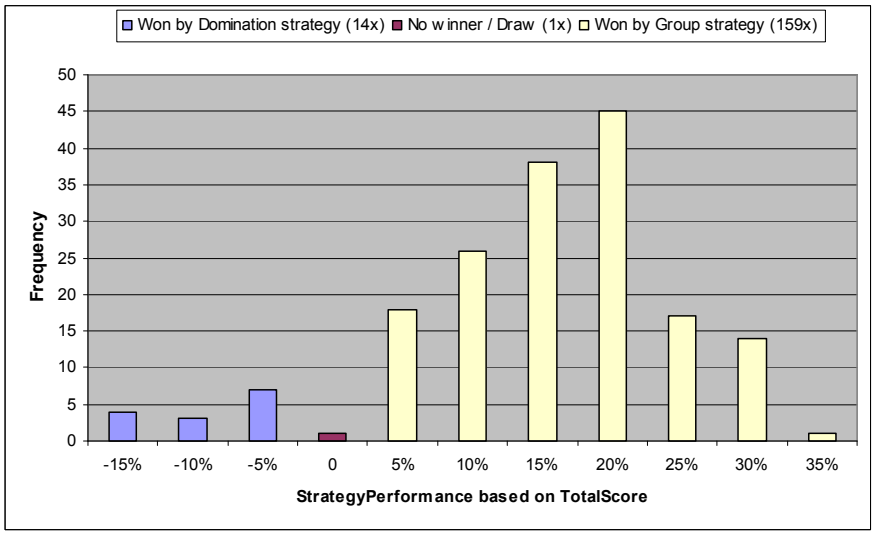

Figure 21: Overview of the StrategyPerformance results of the 174 test games to compare the Domination and new Group strategy.

Figure 21 shows the StrategyPerformance of all 174 test games. Since both strategies have a certain amount of randomness in them, it may happen by chance that one strategy performs a better target selection than the other, resulting in a stronger position for the first strategy. The game's physics simulation adds another fuzzy factor, as its results are not constant over multiple runs of the game. These factors however, cannot account for the difference in the number of games that was won by either side. The Domination strategy managed to win 14 games, while the Group strategy managed to win 159. It also shows that for about half (77) of those 159 games, the Group strategy managed to achieve 'a greater victory' than the Domination strategy achieved in any the games it won.

6.3 System extensibility

This section discusses the extensibility of the multiplayer bot system. Section 6.3.1 describes how the current implementation of the Group strategy for Capture and Hold may be extended to allow for more variation in bot behavior. Section 6.3.2 discusses which parts of the code are generic and which parts would need to be changed or updated if a new game mode were to be supported by the AI.

6.3.1 Extending the AI for Capture and Hold

Section 5.1.1 described how the Strategy Parameter Manager of the Group strategy determines the number of desired captured areas. Any change in this component results in very noticeable changes in the strategies. The default Capture and Hold strategy always subdivides the bots into smaller groups. Depending on how many areas are targeted, squads are formed and scouts are sent to harass bases that are

not otherwise selected as targets. By fixing the desired number of areas to one, and disabling the scouts, the bots become much more focused. The bots do not split up anymore, but instead stay together in one large group. This provides a strong offensive force. Figure 19 shows a test game between the Group and the Mass strategy in progress. To increase the variation in the bots' strategy, the Strategy Parameter Manager could be extended to dynamically change the desired number of areas.

6.3.2 Adding AI for another game mode

To add a strategy for a new game mode, part of the code can be reused. This section discusses which parts of the code could be reused and which parts should be rewritten or changed if the multiplayer bot system were to be extended for use with the game mode capture the flag (see section 1.2.1).

As first mentioned in section 5.1, part of the faction leader's logic is programmed in a base class, intended for reuse by other strategies for the same or other game modes. The base class contains code for squad management, allowing for easy creation, destruction and tracking of squads that command multiplayer bots. The squad management also associates a strategy with the type of squad that it uses. For example, the Group strategy uses squads that have a different HTN domain than the squads used by the Domination strategy.

The faction leader base includes pair wise functions to retrieve the faction (needed for example to determine which Capture and Hold areas are captured), the number of completed / captured objectives, the mission score and the total number of objectives, each for the friendly and enemy faction.

Furthermore, the faction leader base defines the interface that attaches the faction leader to the game code. Each implementation of a faction leader must specify an update function. For the Capture and Hold Group strategy implementation, this function updates the faction leader's three main components, as discussed in section 5.1. The interface does not contain separate entries for these three components, as implementations are not expected to use identical update sequences for these components. Another mandatory function is the spawn point selector as discussed in section 5.1.4. Handling squad messages is optional, as not all strategies would require squads to communicate to the faction leader directly.

Capture the flag focuses around two objectives; the enemy flag, which needs to be captured and brought to the home base and the friendly flag which needs to be guarded or recaptured when stolen by the enemy. The Strategy Parameter Manager would in this case divide the total number of bots over the two objectives. The Mission Strategy Executive would assign all bots to attacking or defending squads. The callbacks described in section 5.1.4 need a new implementation as both the selection of spawn points and the processing of squad messages are game mode specific.

The squad layer also contains parts that may be reused for the implementation of squads for other game modes. The multiplayer squad daemon described in section 5.2.1 provides game mode independent information that most multiplayer squads would use. (squad member, level and mission information)

Squads that are using squad states and squad member states as described in section 5.2.2 can reuse a large part of the squad HTN domain. The state transitions as shown in Figure 18 are encoded in two methods, one that controls the squad state, and one that controls the squad members' states. These two methods must be rewritten to support the capturing and defending of flags. The mechanism that resets squad (member) states is located outside these two methods and may remain unchanged.

7 Conclusion

7.1 Objective

The primary objective of this study is to answer the following question: How may the Killzone 2 AI system be used to create an extensible multiplayer bot architecture that supports the development of entertaining multiplayer bot AI which is more challenging than the multiplayer bot AI of Killzone 1? Section 7.1.1 discusses this question, while section 7.1.2 will answer the secondary questions on the use of the HTN planner.

7.1.1 Multiplayer bots for Killzone 2

In chapter 4 and 5 we have discussed the design and implementation of a system capable of controlling multiplayer bots in Killzone 2. The system has been designed with a focus on cooperative behavior. It supports controlling the bots in an hierarchical fashion. On top of the hierarchy, there is one coordinating process called the faction leader, which divides the bots in smaller groups. To accomplish the goals set by the rules of the multiplayer game mode, the faction leader sends orders to these groups. Each of these groups is coordinated by group (or squad-) leader which receives orders from the faction leader and delegates them by sending orders to group members.

To ensure extensibility of the system, both the faction leader and the group leader have been designed and implemented in a game mode independent way (see section 7.3). This allows for future support of other game modes. We have shown by example that the system supports the creation of bots for the multiplayer game mode Capture and Hold (described in section 1.2.4).

The implemented strategy for Capture and Hold has been tested against human players and against a translated version of Killzone's multiplayer bot AI to assess the entertainment value and difficulty of the bots. (see section 7.3 for a summary). Although it could not be tested if the new AI is more challenging than the AI from Killzone, the new AI is more focused on coordinated behavior, which increases the need for human opponents to coordinate their own actions order to win.

7.1.2 Using the HTN planner for multiplayer bots

As explained in section 3.2 , the HTN planner's main strength is the execution of logical operations, interpreting knowledge domains. The HTN planner is not tailored for numerical operations. Since the faction leader's task requires evaluation of heuristics and the calculation of distances quite often, we have chosen to implement it in C++. The squad leader is much less dependent on numerical operations. It's main task is to coordinate the actions of the squad members given a specific order from the faction leader. Squad leaders are implemented using HTN knowledge domains.

One clear advantage of using the planner to construct squad behavior is the ease of implementing state networks as described in section 5.2.2. The preconditions of the knowledge rules can be used to monitor state changing conditions, while the planner's database may be used to save the current state, including that of all squad members. This technique was used in the domain of the multiplayer squad for Capture and Hold to accomplish an interruptible regroup – attack – defend sequence. (section 5.2.2)

Another advantage of using the planner is that it forces the developer to think about which data is actually needed to make decisions during planning. Data must be explicitly made accessible for use in the domains by creating a daemon that inserts the data into the planner database. This could prevent some cases of cheating AI, where

the developer uses information carelessly, without thinking if the information could actually be deducted by a human player in the same situation as the AI controlled individual.

Although the use of the HTN planner is convenient, the squad behavior could have been implemented without it. The plans constructed by the squad domain consist of sequences of orders and state changes. There are no squad tasks that implement the regroup, attack or defend behavior. Instead the squad domain contains conditions for switching between states, each switch causing new orders to be sent to squad members. Section 7.4 describes a possible change in the implementation of the squad leader that would use the HTN planner more as a planner, constructing sequences of squad actions.

Besides not handling numerical operations elegantly, the HTN planner is inefficient in indexing lists, picking elements at random, or shuffling a list of elements. There are two tasks in the Capture and Hold scenario that required list indexing: squad leaders choosing a regroup location out of several available and the individual soldiers choosing a location within the target capture area. Since these tasks had to be performed only incidental, both tasks were solved by creating a call term, regular C++ code, invoked when needed.

7.2 Deliverables

In chapter 4 and 5 we have discussed the design and implementation of both deliverables. The first deliverable, the *generic structure within the game's AI system that enables AI developers to implement behavior for each multiplayer game mode* is delivered by:

- The functionality and the interface of the faction leader base (section 4.2 and 5.1) providing multiplayer squad management and communication to and from squad leaders.
- The multiplayer squad domain structure allowing for the creation of 'state driven' squad member control (section 5.2.2) and the mission independent part of the multiplayer squad daemon (section 5.2.1) providing squad leaders with information on all multiplayer bot squad members.

The second deliverable, *a single application of this system, a strategy for one of the game modes* is delivered by:

- The Capture and Hold specific implementation of the faction leader, providing target selection, spawn point selection, a division of forces. (section 5.1)
- The methods in the multiplayer squad domain that define the squad state and squad member state changes (section 5.2.2) and the daemon providing squad leaders with information on Capture and Hold areas. (section 5.2.1)

7.3 Evaluation

Section 6.1 describes the setup and the results of a series of test rounds performed with 8 bots on one side and a team of 8 human players on the other. Participants were Guerrilla employees that play many multiplayer games and know the multiplayer bots of other games. All 24 participants were asked to provide feedback on the basis of a number of questions. The respondents' feedback was used to evaluate the bots' difficulty and entertainment value. Respondents replied that they found the bots' difficulty to be "challenging", "about right", "tough but beatable". Two participants

thought the bots were “hard to beat”. (see section 6.1.3) On this matter, the bots exceeded the requirement *challenging for novice players* as the testers were not novice players. (They were all employees of Guerrilla.)

Unfortunately, most respondents replied that they did not think that the bots showed human like behavior or intelligence. Their behavior was characterized as “robotic” and “slightly repetitive/predictable”. Section 7.4 discusses probable causes and possible solutions / improvements.

Nevertheless, the fact that the bots noticeably coordinated their actions was appreciated. (*“I loved how the bots tried to do stuff together! As a group!”* – design/2) Almost all respondents noticed the coordinated behavior of the bots, e.g. bots moving, attacking and / or defending together. It seemed to increase their own tendency to cooperate. During several test rounds, the members of the human team started communicating – shouting orders across the office – in order to coordinate their actions. Ten of the eighteen respondents explicitly mentioned that they enjoyed testing the bots, calling them *“[very/good] fun”*.

The matter of extensibility of the multiplayer bot system has been discussed in section 6.3. We have shown which parts of the multiplayer bot system are game mode independent and may be reused and which parts need a new implementation in order to construct AI for another game mode. (also see section 7.2).

The last evaluation criterion says that the bots should have an *improved strategy over Killzone's AI* in terms of cooperation and compared to the Domination strategy. Section 6.2 describes how Killzone’s Domination strategy was ported to the new multiplayer bot system, how the two strategies were compared and what the results of this comparison were. Before the comparison was made, Killzone’s Domination strategy was made up to date with the changes in the Capture and Hold game mode. This way we have shown that the grouped attack and defense of the new strategy outmatches the Domination strategy.

7.4 Future work

7.4.1 System improvements

As mentioned before, an important improvement to the multiplayer bot system is the support for other game modes. Besides the implementation of specialized faction and squad leaders, some game modes may require some extra work to support multiplayer specific individual actions, such as placing explosives, which will be needed for any Assault type game modes. (see section 1.2.1)

One change that could lead to a better use of the HTN planner for squad control would be to make a series of generic squad tasks (move, attack enemy, place explosives, escort friendly, capture area etc...) and to define all squad behaviors in terms of these tasks instead of using the state based squad member ordering approach described in section 5.2.2. An advantage to this approach is that the plans resulting from the domain would reflect the squad’s course of action at any time as it does for individuals. A disadvantage is that squad member messages would need to be processed in a separate way, as the squad leader would not be able to temporarily suspend its current task in order to process these messages. The planner does not allow switching between two active plans (multitasking).

7.4.2 Better use of cover

During the evaluation by human test teams, some participants characterized the behavior of the bots as robotic (as opposed to human like). One of the causes of this characterization has been the bots' limited use of cover and their limited reaction to being hit. As described in section 5.2, squad leaders use the highest priority orders to command the squad members, which causes squad members to consider the execution of the order as more important than stopping to take cover or to fire at an enemy. Ideally, when the bots are attacked while they are on the move, some of the group members would seek cover and try to disable the enemy, while others continue to move. A similar solution could work for attacking squads. Some rush in the Capture and Hold area to neutralize it, while others are more careful and attack from behind cover.

When defending a Capture and Hold area, bots would best seek cover from the directions from which enemies are likely to appear, perhaps changing their position from time to time. Some of the defendants could use a hiding location, keeping itself out of sight of enemies until they try to attack.

7.4.3 Squad movement

Another reason for the robotic appearance of the bots is the fact that groups of bots do not spread out while on the move. The way the squads move was said to be "ant like", as all squad members choose identical paths and end up walking directly behind each other in a long line. To overcome this, the squad members' movements should be coordinated, creating more space between squad members, possibly in a squad formation.

The regrouping of the bots caused the bots that already arrived at the designated location to stand still (almost) completely. A well thrown grenade could wipe out almost the entire squad. (Which is why grenades were taken out for the play tests.) Since any novice FPS player will say that it is best not to stand still, the current regrouping not only makes the bots less strategic, but also less human like. A solution may be found in bots that gather while being on the move, or bots that move around while waiting for others to join.

7.4.4 More variety in strategies

One way to make the bots more fun and / or challenging is the addition of a number of different strategies for the same game mode (including a number of opening strategies) plus a way to select or rotate them. As one of the participants of the test rounds suggested: "Define a couple of strategies and mash them up so it's harder to know what to expect." Which kind of strategies may be applied partly depends on the number of available bots. A simple way to add more variety to the Capture and Hold bots by manipulating the number of desired captured areas was discussed in section 6.3.1.

7.4.5 Improved use of game feedback

Human players do not just rotate their strategy at random. (At least, not the better players.). They take into account how well different strategies seem to be working. The same holds for target and spawn location selection. Human players observe when a certain approach does not work and change their course of action accordingly. The target selection of the Capture and Hold strategy is adaptive to some extent, as described in section 5.1.2. It directly influences the preference of a Capture and Hold

area based on how often the squad that has targeted that area is attacked. An unwanted side effect is that Capture and Hold areas are also lowered in preference when they are attacked often, regardless of the outcome of the defense. Instead, areas that are easily defended should be preferred, even when they are often attacked.

Additional information could be used to change strategy or tactics. For example, the number of casualties versus the number of enemy casualties in some specific area / location could indicate a danger level for an area. Position information on enemy units (for as far as they are known) may give a clue on which area's on the map are less well defended. For example, when a bot squad is defeated by 5 human players nearby an area on one side of the map, areas on the other side of the map are likely to be easier to capture. These types of information could be used not only in target selection, but also in path planning and spawn location selection (taking a safe / sneaky route).

7.4.6 Level dependent tactics / strategies

Since human players retain tactical knowledge about the multiplayer levels they play, the bots may be equipped with knowledge about which strategies work best in which level, and which of the targets should be considered more important than others without this being considered cheating. For the Southern Hills level for example (see overview in Figure 19) the middle base is of greater importance than the other two bases. Besides the initial preferences for certain targets, level specific information could include the location of entry points aka choke points for each objective. Choke points define the defense perimeter of an objective. For example, the middle Capture and Hold area of Southern Hills is located in a bunker that has three entries. If these three points are secured, no enemy unit can approach the objective unnoticed.

8 References

[1] Beek, JP van der (2007), MSc Thesis, *Autonomous Squad Behavior in a Virtual Game Environment*, Vrije Universiteit Amsterdam, Faculty of Sciences.

[2] Bratko, I. (1990), "Matching", *Prolog Programming for Artificial Intelligence*, 2nd ed. (pp 38-43), Harlow: Addison-Wesley.

[3] Charniak, E., Riesbeck, C.K., McDermott, D.V., Meehan, J.R. (1987), "Lisp Review", *Artificial Intelligence Programming*, 2nd ed. (pp 1-33), Hillsdale, NJ: Lawrence Erlbaum Associates.

[4] Erol, K., Hendler, J. and Nau, D.S. (1994). *HTN Planning: Complexity and Expressivity*, Proceedings of the twelfth national conference on Artificial intelligence (vol. 2) (pp. 1123-1128), Seattle, WA: American Association for Artificial Intelligence.

[5] *F.E.A.R.*, In Wikipedia, The Free Encyclopedia. Retrieved July 4th, 2007, from http://en.wikipedia.org/wiki/F.E.A.R.

[6] *Finite state machine*, In Wikipedia, The Free Encyclopedia. Retrieved May 7, 2007, from http://en.wikipedia.org/wiki/Finite_state_machine

[7] *First-person shooter*, In Wikipedia, The Free Encyclopedia. Retrieved August 12, 2007, from http://en.wikipedia.org/wiki/First_person_shooter

[8] Gilgenbach M. (2006), *Fun Game AI Design for Beginners*. In S. Rabin (Ed.), *AI Game Programming Wisdom 3* (pp 55-63), Hingham: Charles River Media Inc.

[9] *Guerrilla Company Profile*, Retrieved July 8, 2007, from http://guerrilla-games.com/company_profile.html

[10] *Guerrilla Games*, In Wikipedia, The Free Encyclopedia. Retrieved July 8, 2007, from http://en.wikipedia.org/wiki/Guerrilla_Games

[11] *Halo 2*, In Wikipedia, The Free Encyclopedia. Retrieved July 5th, 2007, from http://en.wikipedia.org/wiki/Halo_2

[12] Higgins D. (2002). *Generic A* Pathfinding*. In S. Rabin (Ed.), *AI Game Programming Wisdom*, (section 3.2, pp. 114-121), Hingham: Charles River Media Inc.

[13] Hoang, H. (2005), MSc Thesis, *Using HTN to Coordinate Unreal Tournament Bots*, Lehigh University, USA

[14] Ilghami, O. (2006). *Documentation for JSHOP2*. Retrieved March 18, 2007, from http://www.cs.umd.edu/~okhtay/jshop2doc.pdf

[15] Isla D., Blumberg B. (2002), *Blackboard Architectures*. In S. Rabin (Ed.), *AI Game Programming Wisdom*, (section 7.1, pp. 333-344), Hingham: Charles River Media Inc.

[16] Isla, D. & Blumberg B. (2002). *Blackboard Architectures*. In S. Rabin (Ed.), *AI Game Programming Wisdom* (pp 333-344), Hingham: Charles River Media, Inc.

[17] Isla, D. (2005). *Handling Complexity in the Halo 2 AI*, Game Developers Conference 2005 Proceedings

[18] *JavaBot for Unreal Tournament*, an open source project on Sourceforge. Retrieved April 7, 2007, from http://utbot.sourceforge.net/

[19] *Killzone manual (Vekta Today)*, as found in the PAL version of Killzone, Developed by Guerrilla, Published by Sony Computer Entertainment Europe, 2004

[20] Laird, John E. (2000), *It knows what you're going to do: Adding anticipation to a Quakebot*, AAAI 2000 Spring Symposium on Artificial Intelligence and Interactive Entertainment.

[21] Luger G.F., Stubblefield W.A. (1998), Artificial Intelligence, 3rd edition, chapter 13, *Machine Learing: Symbol Based* (pp. 603-660). Addison Wesley.

[22] Luger G.F., Stubblefield W.A. (1998), Artificial Intelligence, 3rd edition, section 15.3.2, *Evolutionary Programming* (pp. 740-743). Addison Wesley.
[23] Luger G.F., Stubblefield W.A. (1998), Artificial Intelligence, 3rd edition, chapter 14, *Machine Learning: Connectionist* (pp. 661-712). Addison Wesley.
[24] Luger G.F., Stubblefield W.A. (1998), Artificial Intelligence, 3rd edition, section 4.2, *Admissibility, Monotonicity, and Informedness* (pp. 139-144). Addison Wesley.
[25] *Multiplayer game*, In Wikipedia, The Free Encyclopedia. Retrieved April 7, 2007, from http://en.wikipedia.org/wiki/Multiplayer_game
[26] Nau, D., Cao, Y., Lotem, A. and Muñoz-Avila, H. (1999). *SHOP: Simple Hierarchical Ordered Planner*, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (pp. 968-973), San Francisco, CA: Morgan Kaufmann Publishers.
[27] *No one lives forever*, In Wikipedia, The Free Encyclopedia. Retrieved July 4th, 2007, from http://en.wikipedia.org/wiki/No_One_Lives_Forever
[28] Orkin J. (2004), *Applying GOAP to Games*, In S. Rabin (Ed.), *AI Game Programming Wisdom 2* (section 3.4, pp. 217-227), Hingham: Charles River Media Inc.
[29] Orkin J. (2004), *Simple Techniques for Coordinated Behavior*. In S. Rabin (Ed.), *AI Game Programming Wisdom 2,* (section 3.2, pp. 199-206), Hingham: Charles River Media Inc.
[30] Orkin J. (2006), *Three states and a Plan: The A.I. of F.E.A.R.* Game Developers Conference 2006 Proceedings.
[31] *Quake II*, In Wikipedia, The Free Encyclopedia. Retrieved July 7th, 2007, from http://en.wikipedia.org/wiki/Quake_II
[32] *Quake III Arena*, In Wikipedia, The Free Encyclopedia. Retrieved July 7, 2007, from http://en.wikipedia.org/wiki/Quake_III_Arena
[33] Reynolds J. (2004). *Team Member AI in an FPS*. In S. Rabin (Ed.), AI Game Programming Wisdom 2 (section 3.3, pp. 207-215), Hingham: Charles River Media Inc.
[34] *RoboCup Official Site*, retrieved May 7, 2007, from http://www.robocup.org/
[35] *Simulation*, In Wikipedia, The Free Encyclopedia. Retrieved August 12, 2007, http://en.wikipedia.org/wiki/Simulation
[36] *Soar (cognitive architecture)*, In Wikipedia, The Free Encyclopedia. Retrieved April 2, 2007, from http://en.wikipedia.org/wiki/Soar_%28cognitive_architecture%29
[37] Sterren, W. van der (2002). *Squad Tactics: Planned Maneuvers*, In S. Rabin (Ed.), , *AI Game Programming Wisdom*, (section 5.4, pp. 247-259), Hingham: Charles River Media Inc.
[38] Sterren, W. van der (2002). *Squad Tactics: Team AI and Emergent Maneuvers*. In S. Rabin (Ed.), *AI Game Programming Wisdom*, (section 5.3, pp. 233-246), Hingham: Charles River Media Inc.
[39] Straatman R., Beij. A., van der Sterren, W. (2006). *Dynamic Tactical Position Evaluation*. In S. Rabin (Ed.), *AI Game Programming Wisdom 3*, (section 5.2, pp. 389-403), Hingham: Charles River Media Inc.
[40] Straatman, R. & van der Sterren, W. & Beij, A (2005). *Killzone's AI: dynamic procedural combat tactics*, Proceedings of the Game Developers Conference 2005
[41] *Unreal Tournament*, In Wikipedia, The Free Encyclopedia. Retrieved July 7th, 2007, from http://en.wikipedia.org/wiki/Unreal_Tournament

[42] Wallace N. (2004). *Hierarchical Planning in Dynamic Worlds.* In S. Rabin (Ed.), *AI Game Programming Wisdom 2*, (section 3.5, pp. 229-236), Hingham: Charles River Media Inc.
[43] Waveren, J.M.P. van (2001). MSc Thesis, *The Quake III Arena Bot*, University of Technology Delft, The Netherlands