Behaviour Oriented Design for Real-Time-Strategy Games

An Approach on Iterative Development for STARCRAFT AI

Swen Gaudl
Department of Computer Science
University of Bath
BA2 7AY, Bath, UK
s.e.gaudl@bath.ac.uk

Simon Davies
simon.davies@bath.edu

Joanna J. Bryson
Department of Computer Science
University of Bath
BA2 7AY, Bath, UK
joanna.bryson@bath.ac.uk

ABSTRACT

Design is an essential part of all games and narratives, yet designing and implementing believable game strategies can be time consuming and error-prone. This motivates the development and application of systems AI methodologies. Here we demonstrate for the first time the iterative development of agent behaviour for a real-time strategy game (here StarCraft) utilising Behaviour Oriented Design (BOD) [7]. BOD provides focus on the robust creation and easy adjustment of modular and hierarchical cognitive agents. We demonstrate BOD's usage in creating an AI capable of playing the StarCraft character the Zerg hive mind, and document its performance against a variety of opponent AI systems. In describing our tool-driven development process, we also describe the new Abode IDE, provide a brief literature review situating BOD in the AI game literature, and propose possible future work.

General Terms

Design,Agent Systems, Simulation, Tool-Driven Development, Behaviour Oriented Design

1. INTRODUCTION

Digital games are a significant part of contemporary life and a substantial part of many economies. They also offer stable and versatile environments in which to develop artificial intelligence, offering great potential for researching human behaviour, intelligent systems, and agents in general [23]. Here we focus on the AI development, design and planning for an agent usable in real-time strategy games. Game AI design is typically a complicated process where game designers and programmers aim at the creation of a robust, challenging, believable subsystem of the game that behaves in a way that is plausible to the end user, the player. To address the issue of creating a robust agent in challenging dynamic environments, we have employed the AI development methodology Behaviour Oriented Design (BOD) [7] to implement a real-time strategy AI. We implemented this pilot case study using and extending a new version of the Advanced Behaviour Oriented Design Environment (Abode) [8], which we also present in this paper.

Although originally designed for cognitive robotics, BOD has most commonly seen use in first-person action (FPS) games [27, 4, 17, 16, 19]. Here we present the first application of BOD in the field of real-time strategy. Due to the different nature of first-person action games and strategy games, different problems are present. In first-person action games the focus is primarily on recognising and responding quickly to opportunities and threats, which may seem a more obvious application for reactive AI. In strategy games the focus lies mostly on extended strategic planning, so they are seen as more "cognitive" than FPS. To prove the feasibility of using BOD for strategy games we designed and implemented a game AI using BOD for the game StarCraft. The pilot work presented here cannot claim to present the current leading strategy. Our focus here is on the presentation of the approach itself, translating the understanding of ordinary developers into game AI, rather than focusing solely on the outcome. Nevertheless, performance we report is quite respectable.

Real-Time strategy games (RTS) offer an interesting perspective for AI researchers because they originate from complex board games such as Chess or Go. Most RTS games have fairly simple underlying rules which are easy to learn. During game play similar strategic concepts found also in Chess and Go such as *openings* and *counter-strategies* are used by the players to find and exploit the search for an optimal winning strategy. In contrast to Chess or Go, RTS games offer a greater variety of choices making it even harden to write robust and believable AI. Because BOD has previously been successfully applied to FPS games which are continuous as well but require a different kind of planning, it seemed a logical next step to apply BOD to RTS games to strengthen our understanding of AI planning for games.

This paper is organised as follows. As a starting point a short review of different approaches to, and techniques and strategies for AI development is presented. This is followed by an introduction to BOD and its underlying concepts. After the introduction of BOD the implementation of the game AI is presented including discussions on the gained experiences while developing the StarCraft bot. Implementa-

tion details, as well as results and analysis of the bot are given to illustrate the design and implementation. Finally, we conclude with a description of our case study results and suggest directions for further work.

2. GAME DESIGN TECHNIQUES AND BOD

There are many different techniques that have been proposed and developed for AI applications. In the presented project one such approach, BOD, is used and evaluated in the context of real-time strategy games. In this section we introduce BOD in the context of related AI techniques both to provide context and to elucidate the understanding of the AI concepts underlying the BOD approach. This will cover both general AI and more specifically approaches that have been used for StarCraft or similar games.

2.1 Finite State Machines

One of the first methods for AIs that was developed and is still fairly common is the *finite state machine* (FSM). It is an abstraction of a system represented through states and a set of conditions where changes in the system are represented by transitions between them. FSMs have been used for game AI since some of the earliest games, eg. Namco's Pac-Man [2, p. 165]. FSMs are suited for small purely reactive systems or ideally subsystems in a larger AI system, as they do not support planning and prediction but offer an intuitive logic representation. A perfect subsystem inside StarCraft would be controlling of the different states of a single unit.

2.2 Potential Fields

For the control of individual units, one method that is currently popular is *potential fields*, also known as *potential functions* [2, p. 80]. A potential function can be mapped to entities within the world and gives them either a repulsive or attractive force [26]. In the case of game AI, this can be used to attract units to weak and desirable targets while repelling them from stronger more dangerous forces. With potential fields, a single function can be used to provide a large amount of control. By modifying the variables of the repelling and attraction forces a way can be provided to include learning and improvement into an AI mechanism. This has been done by AI's such as the Berkeley Overmind to great effect [20]. There are also approaches covering evolutionary multi-agent potential fields (EMAPF) in small controlled training examples [28].

2.3 Machine Learning and Planning

For most naive developers, the expectation for AI is that it should fully replace the design with automated search approaches such as machine learning or planning. For example the *neural network* approach attempts to replicate the neural systems of the brain. The most prominent case of a still-active neural network use inside a deployed commercial game is the *creature* in Black&White2. Other games such as *Creatures* [18] evolve neural networks for control and learning in the virtual world. A more general-purpose type of machine learning approach called *reinforcement learning* has been used for developing StarCraft AIs [24], and has been proven to be effective way to control groups of units. However, all forms of machine learning still require careful design in order to reduce the combinatorial possibilities of the behaviour to be learned to something tractable. This fact is sometimes overlooked or under-reported in the description of the algorithms.

Search Algorithms (SA) such as Monte Carlo Tree Search (MCTS) [6] have seen recent success in games like Go and even in games with incomplete information such as Magic: The Gathering [30]. SA offer great potential in exploring a huge search space, but again require careful problem specification. The main difficulty when trying to use search in current commercial digital games such as *StarCraft* is, as with machine learning, the search space is too big. There is also normally no parallel simulation of sufficient play-throughs of a complete game possible [3] which is necessary to allow for a decently-skilled artificial player.

2.4 Goal-Driven Autonomy

An approach to creating an AI model that can allow automated planning to be tractable is using *Goal-Driven Autonomy* as a conceptual model [31]. This approach has been used in the context of StarCraft. The key components of goal-driven autonomy are a planner module that generates plans to reach a goal, and another module that can detect discrepancies from the expected outcome and the actual outcome, see figure 1. Explanations for these discrepancies are then used to generate new goals which are then sent back to the planner. This approach was found to be useful for the high level strategies in StarCraft, where the system would have an overall plan. A goal oriented approach has many useful features that make this approach a large improvement over less complex approaches such as finite state machines. The ability for plans to dynamically be adjusted based on the discrepancies in the desired outcome and the actual outcome of events proves to be a powerful feature in the case of a strategy game.

2.5 Behaviour-Based Artificial Intelligence

Building upon the ideas of a finite state machine is a technique known as *Behaviour-Based Artificial Intelligence*, developed by Brooks [5]. This approach adds modularity to a FSM system, by having multiple finite state machines each specialised to a sub task of the problem and thus easier to program. In the original BBAI methodology, the *subsumption architecture*, FSM can only have interactions with one another by modifying the inputs and outputs of the other FSM / modules. Different layers of control are built into a simple stack hierarchy, where the higher layers can subsume the roles of the lower level modules by modifying their outputs appropriately. This is the idea behind a BBAI where an AI is built bottom-up with the overall goal on the highest layer of the system. The original proposition of BBAI made no inclusions for machine learning or planning; In this form it is another purely reactive architecture, however one much more scalable to large systems compared to a basic FSM. However, later BBAI used other representations for the modules than FSM, including potential fields [22, 1].

2.6 Behaviour Oriented Design (BOD)

BOD is an AI development methodology that combines an iterative development process based around variable state for learning and planning with the modular, responsive architecture of BBAI. It emphasises producing extensible and

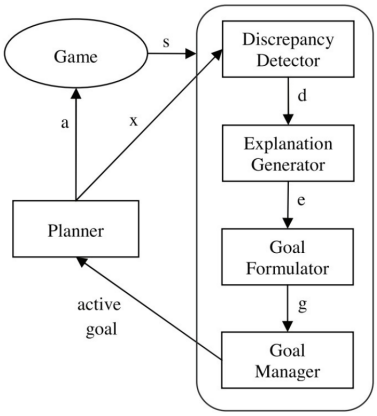

Figure 1: Components in the simplified Goal-Driven Autonomy conceptual model [31]

maintainable AI projects. First introduced in 2001 [7], the approach emphasises the *design* of AI systems allowing and facilitating strong human design, which in turn facilitates the application of search techniques like learning and planning by focussing them on only the representations and problems these algorithms can reliably and efficiently solve. Unlike approaches that attempt to use single representations for all behaviours (e.g. potential fields or FSM [1, 5]), BOD encourages the designer to utilize a variety of representations to enrich the designed behaviour. Facilitating human design is important because in most AI systems, human designers are still required to do most of the hard work of coping with the combinatorial explosion of available options [9]. The iterative design heuristics encourage developers to find a more intuitive and better way to solve a problem, whether with their own coding or by inserting learning or even off-the-shelf AI components.

```
/** Builds a pair of zerglings from any larvae**/
public boolean spawnZergling(){
 for (Unit unit : bwapi.getMyUnits()) {
  if (unit.getTypeID() ==
       UnitTypes.Zerg_Larva.ordinal()) {
   if (resourceManager.getMineralCount() >= 50 &&
        resourceManager.getSupplyAvailable() >= 1
       && buildingManager.hasSpawningPool(true)){
    bwapi.morph(unit.getID(),
         UnitTypes.Zerg_Zergling.ordinal());
    return true;
   }
  }
 }
 return false;
}
```

Listing 1: The primitive action "spawnZergling" which is implemented in Java. Such primitives are referenced by the POSH plan which determines when they should be executed.

To determine the priorities between modules and thus to specify when an action should be performed in a BOD system, an *action plan*, also known as a *reactive plan* is used. These are a prioritised set of conditions and the related actions that should be performed when these conditions are met. The action selection mechanism used in BOD is ordinarily a *Parallel-rooted Ordered Slip-stack Hierarchical* (POSH) reactive plan. POSH plans, first described in [] build upon the idea of *Basic Reactive Plans (BRP)* which are a collection of *production rules,* arranged hierarchically according to subtasks.

There are thus two parts to the BOD architecture:

- **Behaviour Libraries** These are primitives — actions and senses that can be called by the action selection mechanism, and any associated state or memory required to support these. They are created in the native language for the problem space, see Listing 1. They determine *how* to do something. The separation of behaviour libraries from the action selection is important. This faciliates code reuse, introducing the possibility of a pool of complex specialized approaches and techniques and at the same time hiding this the underlying structure from the plan itself.
- **POSH Action Selection** Plans in contrast specify the particular priorities and even personality of a given character. The POSH action selection mechanism consists of plans which are a hierarchy of actions with associated triggers. Those triggers determine *when* to perform an action. A POSH plan is split up into a *drive collection*, *competences* and *action patterns*. See Figure 2 for a basic plan using the primitive action *spawnZergling* which will be active as soon as a *SpawningPool* — the building which creates those units — is available and enough supply remains after spawning those units.

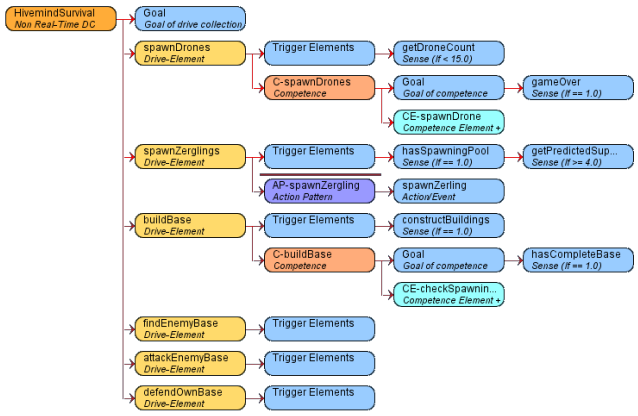

Figure 2: Initial POSH plan from inside the Abode Editor for the Zerg Hivemind which builds drones, mines Crystal and builds Zerlings after creating a *SpawningPool*.

Besides its archictecture, BOD also consists of a set of heuristics for refactoring between these representations and maintaining the clearest model possible for the AI character [10]. BOD has previously been used in the context of writing game AI by Partington, in which an evaluation of BOD principles are performed in the context of writing an AI for the first person shooter title Unreal Tournament [27]. Further details on BOD and the development process are given below.

2.7 Behaviour Trees

The behaviour tree is another type of AI design methodology based on reactive planning that has over the past decade come into a fairly widespread use within the games industry. Due to a similar underlying planning concept, BTs display several noticeable similarities to BOD. An AI development technique first used in an commercial product by the Bungie games studio with the release of HALO2 [21], BTs have since been used in titles such as SPORE, CRYSIS as well as being used again by Bungie in the game HALO3. Unfortunately, there are few academic publications on behaviour trees, but there are several industry articles [13, 14, 25] as well as conference talks [12]. Much of the information about behaviour trees in this document comes from industry resources and is centred around the approach established by Champandard [14]. It is worth noting that different companies have taken different approaches in using BTs. However the basic concept of all those implementations is a modular decomposition of actions into a hierarchical tree, much like POSH. The iterative development methodology core to BOD can in principle be applied to any hierarchical reactive planning system, including BTs; however, here we have used POSH.

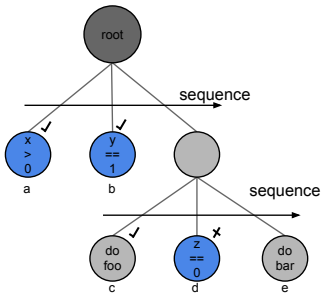

Figure 3: Behaviour Tree featuring two sequence nodes. The Tree is traversed top to bottom, left to right creating a priority order for horizontal traversal and a vertical hierarchical structure.

An example behaviour tree can be found in Figure 3. In this behaviour tree, the root node is a sequence with three children, two conditionals and a sequence node.

3. CASE STUDY: BOD APPLIED TO REAL-TIME STRATEGY GAMES

The previous section positioned BOD inside the field of other AI approaches commonly used for game AI. While automated planning and machine learning techniques can be powerful tools, they cannot solve large problems in tractable amounts of time. A design methodology focussing on hand-programmed plans allows the developer greater control over the resulting behaviour of an AI system. Behaviour trees do not have a standardised complete implementation like POSH does, or arguably POSH is the only freely-availabe and fully-described form of BT, given the techniques are so close. Automated learning and planning can be used in a modular way to produce actions where useful, while BOD provides high-level structuring and planning.As this work focuses on demonstrating the development of an AI for complex strategic games, we now turn this.

3.1 StarCraft

For our case study we designed a first prototype AI for STARCRAFT to play the hive mind of the Zerg, and played it against a variety of other AI systems including the one the final game was shipped with and several available AIs over the Internet. STARCRAFT, first released in 1998 focusses on strategic real-time player-versus-player game play in a futuristic setting. Due to the good balance of the available in-game parties the players can choose from, it has become famous in e-sports [29] and attracted significant media attention.

There has been a recent influx of work that has been done in this area, due to the AIIDE and CIG organisations that started using STARCRAFT for AI competitions. The creation of the BWAPI [11] interface has facilitated the creation of artificial agents for STARCRAFT. This has lead to much recent interest in the AI development community over the past few years. The game provides a complex environment with many interaction possibilities which involves players having to manage a multitude of different tasks simultaneously. STARCRAFT also requires players to be able to react to changes in the game world in real-time while in parallel controlling and keeping track of long term goals. The given setting introduces many challenges regarding pro and re-activeness of an agent, planning, and abstraction, modularity and robustness of game AI implementations.

3.2 System Architecture

For the presented work the design made use of the Java Native Interface version of BWAPI (JNIBWAPI). This was chosen because the Java Interface can easily be connected to the current POSH action selection mechanism by running POSH in Jython. This means that Java methods can be called directly from the action selection mechanism, without using an additional wrapper as shown in Figure 2. JNIBWAPI makes use of the capabilities of BWAPI where it acts as a proxy. Here BWAPI runs as a server, and the AI connects to this to control StarCraft. This allows the flexibility to run external applications such as JyPOSH, which is much more difficult with standard BWAPI which simply executes code from .dll files.

The result of using JNIBWAPI to communicate with StarCraft, is that the behaviour libraries are written in Java. Then, these behaviours are called from the POSH action selection mechanism through Python code that is run using Jython. The overall architecture can be seen in Figure 5.

Once the system architecture had been developed, the next step was to create a proof of concept that performed most of the basic tasks required to play a game of StarCraft. To do this, the first goal was to implement an AI that would only build the basic offensive unit, the zergling, as quickly as possible, and then attack. This is a basic strategy that has been used by players to attempt to quickly win games without having to have a long-term strategy, see Figure 6. Once all of the behaviours were implemented and tested, the JNIBWAPI behaviour code was integrated with POSH via Python behaviour modules. These were scripts that contained behaviour wrappers for the Java methods, and is used by POSH to determine which methods are senses or actions. As mentioned earlier these are the interfaces that can be

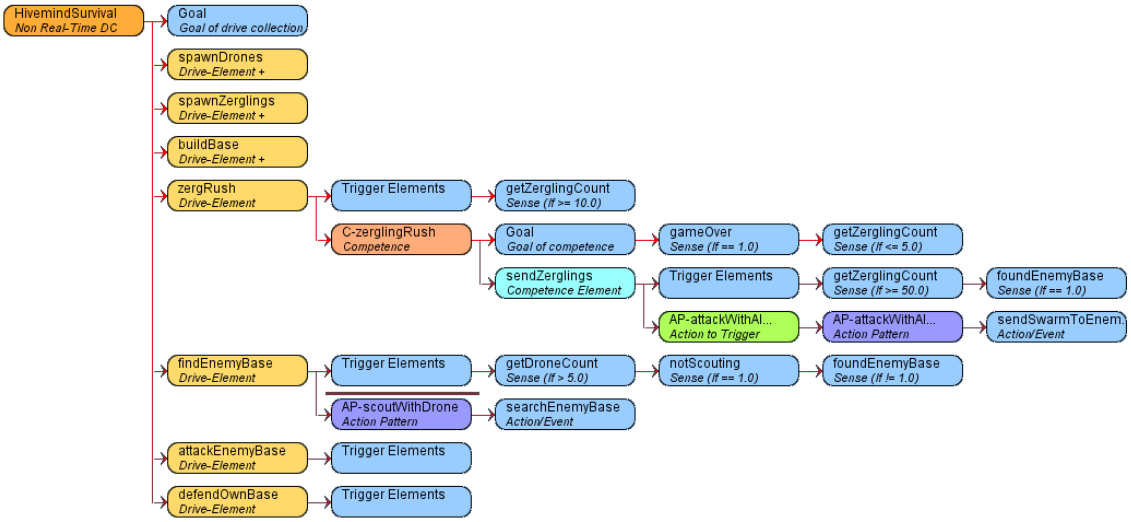

Figure 4: Extending the previous plan from Figure 2 to include the *Zergling-Rush* strategy. It will send a large group of Units to the enemy base as early as possible in the game.

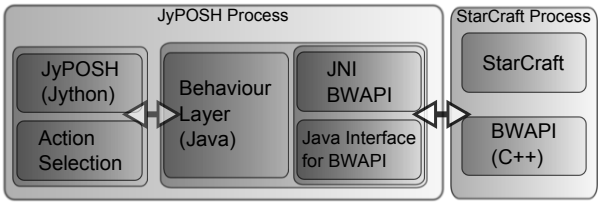

Figure 5: The architecture model for the StarCraft BOD AI. The architecture is separated into three elements the action selection done by POSH on the left. The middle part represents the behaviour primitives, e.g. actions and senses. The rightmost block shows the actual game and the interface which allows the deployment of strategies.

used by POSH plans. The Python code for this project is little more than calls to the Java methods. All of the logic and behaviour for the AI is either in the POSH action plan, or in the Java behaviour modules.

3.3 Iterative Development

The BOD architecture provides developers with a structure and a process to quickly develop an AI system provided that the POSH mechanism can be connected to the behaviour modules. BOD's design methodology and process also informs developers how to approach AI problems [10]. In contrast to the standard waterfall model, development of AI following BOD is an iterative or agile process that focuses on having a flexible set of requirements, and heuristics to re-evaluate the system's decomposition as behaviours and plans are implemented and tested. The ABODE editor which is freely available[1] helps facilitate BOD by providing a graphical tool to edit and create plan files. The plans presented in Figure 2 and 4 were created using it. The editor allows node collapsing which increases the overall readability of the plan, because one can collapse sub-trees which currently do not need to be touched. This allows attention to be concentrated on specific parts of the tree, keeping the structure clear.

Because BOD facilitates the extension of existing behaviours by adding new elements such as being able to use new unit types, complex behaviour can be iteratively developed and step-wise tested. Due to the separation of the behaviour libraries from the POSH plans, the underlying implementation can be adjusted and independently developed creating an interface-like structure. Also, once a behaviour library is developed, different strategies or players can be quickly implemented by shifting priorities within the reactive plan. Further, an advantage of using hierarchical plans is that they are very extensible — new drive elements can be added without any major changes to an existing POSH plan.

In applying BOD the first step is to specify at a high-level what the agent is intended to to. To start with the simplest complete strategy—ZERGLING-RUSH— we have zerglings overwhelm the opponent as early as possible in the game. Based on this more complex strategies can be developed by building on the first one, including re-using its components. The iterative process of starting with a basic strategy is well suited for all complex development processes because the tests and progress of the intended behaviour / outcome can be traced throughout the iterations and unwanted behaviour can be traced back to when it was first included.

The second step is to decompose the strategy into sequences of actions needed to build up the strategy. Next, identify a list of required senses (observations of the game) and actions (commands the agents uses to control the game).

[1]Search for *amoni software*. Links to ABODE, and our Brood War AI are near the top of the Amoni Software page.

Race	# Matches	# Wins	AVG BOD Score	StdDev	AVG Opponent Score	StdDev	AVG Score Difference
Protoss	17	7	19546	35729	43294	18916	-70.75%
Terran	18	12	56651	26077	35696	15380	37.11%
Zerg	15	13	47961	19218	24333	7974	50.64%
Total	50	32	48257	27680	30523	17594	43.56%

Table 1: Results from 50 matches of the Behaviour Oriented Design bot described in Figure 6 against the Blizzard AI set to random race on the Azalea Map.

Top-Down Analysis:

1. To attack using zerglings, the AI has to build a mass of those units first and then attack the opponents base.
   ↪build mass of zerglings

2. To attack the opponents base, the AI has to know its location.
   ↪scout the map

3. To build a zergling, a spawning pool is needed and enough resources has to be available.
   ↪ build spawning pool
   ↪ ensure enough resources

4. To ensure sufficient resources, they have to gathered.
   ↪ send drones to gather resources

It is important to to remember to always aim for the simplest possible approach first. From the top-down analysis two lists of actions (*mineral_count, support_count, larvae_count, has_spawning_pool, know_opponent_base*) and senses (*harvest_resources, build_spawning_pool, scout_map, build_zerglings, attack_enemy*) were identified.

Having derived a number of basic primitives, those now need to clustered in behaviours as is appropriate to their state dependencies. Keeping in mind the basic principles of OOD such as clustering those elements which create classes having a high internal cohesion and a low external one, we iteratively create a behaviour library. In our case it seemed reasonable to create different behaviours for managing structures, building units, combat and exploration & observation. Those behaviours were in our case written in Java, see for example Listing 1. Once the first behaviours have been developed, a POSH action plan is created to execute them. The plan determines when the AI should call each behaviour to perform an action. The POSH action planning mechanism is part of the BOD architecture using "learnable action plan[2]" (.lap) files which have a lisp-like syntax. The POSH plan was implemented using Abode, which provides both editing and visualisation of POSH plans with a variety of different perspectives on the same plan, see Figure 6 for an easy-print version of the complete plan.

New behaviours were later introduced one or two at a time, and then tested, until the plan was robust enough to deal with a full strategy for StarCraft. The iterative process facilitated by BOD allowed us to build a solid and human-understandable foundation for our first strategy. In Figure 4 the extended plan is presented which introduces the *Zerg-Rush* Drive, one of the most prominent early game strategies. The Drive uses the *Zergling-Rush* Competence as a first step. We would like to extend this plan to allow switching between different Rush tactics by including competences for these. These would need to be prioritized differently according to the assessed stage of the game. Mechanisms for varying priorities include code reuse with multiple drive entry points [27] or setting drives to the same priority and having them guarded by mutually exclusive triggers.

Now we have our first simple strategy which reacts accordingly to the information available in the game and sends in periodic time frames swarms of zerglings to the enemy base. This strategy works well against human players once or twice before they realize how to counter it. After testing this plan one will encounter that if the strategy plays against itself an obvious flaw is present — the absence of a defence.

Following this process a very visual and easily traceable design of the bot is introduced which allows the designer more control over the resulting behaviour by increasing the complexity of the aimed behaviour step-wise. The next POSH elements that were developed were those that dealt with the upgrading of units. Then behaviours were added for research, unit production and then attacking with basic units. The final plan that resulted can be found in Figure 6 which creates units when needed, scouts for the opponent, researches unit upgrades and attacks.

As a next step the underlying primitives were independently updated without the need to touch the POSH plan. The plan could be improved separately; adding more behaviours for creating different types of units.

4. RESULTS

After finishing the first stable prototype the StarCraft Bot was tested on the *Azalea*[3] map against the original Bots provided by the game itself playing adversary races (Protoss, Terran and Zerg) in random mode. Our implementation faired reasonably well in the first tests winning 32 out of 50 matches, see Table 1. We then set the BOD bot against the Artificial Intelligence Using Randomness (AIUR)[4] Protoss bot , which proved a harsher competitor, winning 7 out of 10 against our implementation. On analysis, we realised the BOD bot performed well when it was not attacked by a *rush* early in the game, indicating more room for further iterative development for closing the gap between our prototype and other available bots. The major advantage of our approach is the focus on rapid prototyping and short development times allowing for continuous testing of the implementation.

[2]Note this name is misleading — early experiments in learning plans lead to the current emphasis on design.

[4]AIUR:`http://code.google.com/p/aiurproject/`

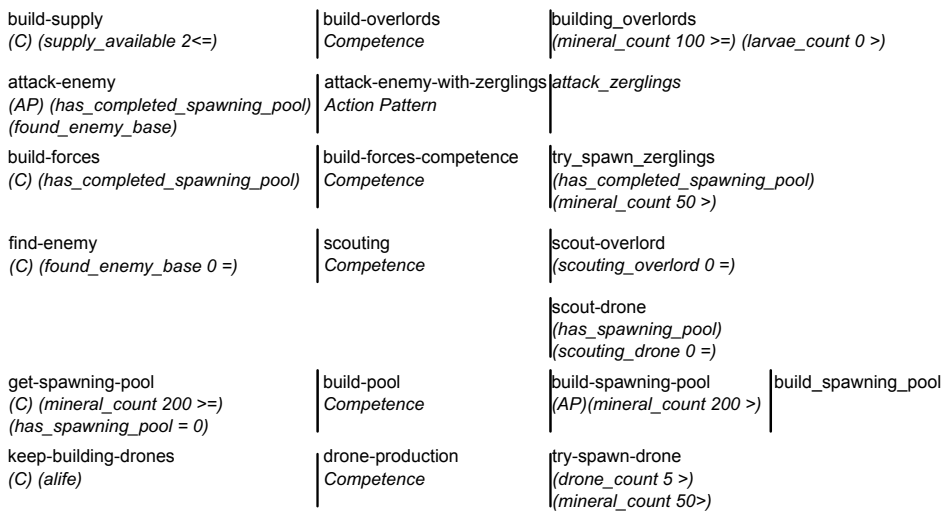

Figure 6: A more complete POSH plan to attack with a units and includes defensive behaviour as well.

5. CONCLUSIONS AND FUTURE WORK

Real-time game character development requires leaving sufficient space for expert human authoring. This underscores the overwhelming importance of systems AI, even where algorithmic AI like machine learning can be exploited to solve regular subproblems. Here we have analysed and discussed a variety of approaches and techniques, most with implementations in existing strategy game environments. After describing Behaviour Oriented Design and POSH, we introduce the iterative implementation of a demonstration prototype AI for a real-time strategy game. The clean and easy-to-grasp AI produced demonstrates a proof of feasibility. The usage of a separation of underlying mechanics and behavioural logic allows independent development of both systems. The visual presentation of the plan itself can be a powerful tool because it offers a different perspective on the behaviour. Using features like node collapsing, the plan editor and visualiser ABODE also minimizes cognitive load when dealing with large plans. The test runs using the original bot for STARCRAFT show good potential — though the developed AI was a prototype representing one of the less-advanced but recognised strategies.

Based on these first results of our prototype further work on the STARCRAFT AI using BOD seems feasible to allow a closer comparison to more advanced strategies and implementation, e.g. [32, 31]. First steps would be the inclusion of a more sophisticated strategy such as the *Mutalisk-Rush* and fixing the identified early-game defence problem.

The development of AI using ABODE provides good graphical feedback and allowed for fast prototyping the higher level behaviour while separating the underlying implementation allowed for a clean cut between design and close-to-engine programming. A complete tutorial on how to test and set up the environment for the AMONI STARCRAFT Bot AI and ABODE[5] is available on the AMONI Website[6].

During this development, connections between BOD and other approaches became apparent. While Behaviour Trees (BT) have not been standardised nor sufficiently described in academic papers, they are widely used and many parallels between BT and POSH were visible. In future work we intend to clarify the differences between both approaches, or to show possibilities of how to subsume one approach by the other and create a new more formalised and accessible approach. It would also be good if ABODE was extended to provide real-time debugging offering the AI designer useful feedback and statistics on the developed plan. Other directions which are clearly visible are the inclusion of lower-level approaches such as potential fields, neural networks or MCTS in the context of BOD behaviours which might benefit from providing the high level POSH control with a greater level of flexibility. This new gained flexibility is essential for future work on modelling interesting artificial players up to a human-like level in behaviour [15].

6. ACKNOWLEDGEMENTS

We would like to thank the University of Bath Department of Computer Science for supporting the ABODE project.

7. REFERENCES

[1] R. C. Arkin. *An Behavior-based Robotics*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[2] D. Bourg and G. Seemann. *AI for Game Developers*. O' Reilly. O'Reilly, first edition edition, 2004.

[3] S. R. K. Branavan, D. Silver, and R. Barzilay. Non-linear monte-carlo search in civilization ii. In T. Walsh, editor, *IJCAI*, pages 2404–2410. IJCAI/AAAI, 2011.

[4] C. Brom, J. Gemrot, M. Bída, O. Burkert, S. Partington, and J. Bryson. Posh tools for game

[4] The map Azalea, which has previously been used in high level professional tournaments is available at: `http://www.teamliquid.net/tlpd/korean/maps/25_Azalea`, checked: 16.July 2012

[5] `http://code.google.com/p/abode-star`

[6] `http://www.cs.bath.ac.uk/ai/AmonI.html`

agent development by students and non-programmers. In *9th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games*, pages 126–135, 2006.

[5] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, 1986.

[6] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

[7] J. Bryson. *Intelligence by design: principles of modularity and coordination for engineering complex adaptive agents*. PhD thesis, Citeseer, 2001.

[8] J. Bryson, S. Gray, J. Nugent, and J. Gemrot. Advanced behavior oriented design environment. http://www.cs.bath.ac.uk/~jjb/web/BOD/abode.html, 2006. [Accessed 27th Apr 2012].

[9] J. Bryson and L. Stein. Modularity and design in reactive intelligence. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 1115–1120, 2001.

[10] J. J. Bryson. The Behavior-Oriented Design of modular agent intelligence. In R. Kowalszyk, J. P. Müller, H. Tianfield, and R. Unland, editors, *Agent Technologies, Infrastructures, Tools, and Applications for e-Services*, pages 61–76. Springer, Berlin, 2003.

[11] BWAPI Development Team. bwapi — an API for interacting with starcraft: Broodwar (1.16.1) - google project hosting. website: `http://code.google.com/p/bwapi/`, 2010. [Accessed 4th Nov 2011].

[12] A. Champandard. Behavior trees for Next-Gen game AI. website: `http://aigamedev.com/open/article/behavior-trees-part1/`, 2007. [Accessed 4th Apr 2012].

[13] A. J. Champandard. *Ai Game Development*. New Riders Publishing, 2003.

[14] A. J. Champandard. Getting started with decision making and control systems. In S. Rabin, editor, *Ai Game Programming Wisdom 4*, volume 4 of *Cengage Learning*, chapter 3 Architecture, pages 257–264. Charles River Media, Inc., 2008.

[15] S. E. Gaudl, K. P. Jantke, and R. Knauf. In search for the human factor in rule based game ai: The grintu evaluation and refinement approach. In H. C. Lane and H. W. Guesgen, editors, *FLAIRS Conference*. AAAI Press, 2009.

[16] J. Gemrot, C. Brom, J. J. Bryson, and M. Bída. How to compare usability of techniques for the specification of virtual agents behavior? An experimental pilot study with human subjects. In M. Beer, C. Brom, V.-W. Soo, and F. Dignum, editors, *Proceedings of the AAMAS 2011 Workshop on the uses of Agents for Education, Games and Simulations*, Taipei, May 2011.

[17] J. Gemrot, R. Kadlec, M. Bída, O. Burkert, R. Píbil, J. Havlíček, L. Zemčák, J. Šimlovič, R. Vansa, M. Štolba, T. Plch, and B. C. Pogamut 3 can assist developers in building ai (not only) for their videogame agents. In *Agents for Games and Simulations*, number 5920 in LNCS, pages 1–15. Springer, 2009.

[18] S. Grand, D. Cliff, and A. Malhotra. Creatures: Artificial life autonomous software agents for home entertainment. In W. L. Johnson, editor, *Proceedings of the First International Conference on Autonomous Agents*, pages 22–29. ACM press, February 1997.

[19] J. Grey and J. J. Bryson. Procedural quests: A focus for agent interaction in role-playing-games. In D. Romano and D. Moffat, editors, *Proceedings of the AISB 2011 Symposium: AI & Games*, pages 3–10, York, April 2011. SSAISB.

[20] H. Huang. Skynet meets the swarm: how the berkeley overmind won the 2010 StarCraft AI competition. website: `http://arstechnica.com/gaming/news/2011/01/skynet-meets-the-swarm-how-the-berkeley-overmind-won-the-2010-starcraft-ai-competition.ars`, 2011. [Accessed 11th Nov 2011].

[21] D. Isla. GDC 2005 proceeding: Handling complexity in the halo 2 AI. website: `http://www.gamasutra.com/view/feature/2250/gdc_2005_proceeding_handling`, 2005. [Accessed 4th Apr 2012].

[22] K. Konolige and K. Myers. The Saphira architecture for autonomous mobile robots. In D. Kortenkamp, R. P. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, chapter 9, pages 211–242. MIT Press, Cambridge, MA, 1998.

[23] J. E. Laird and M. van Lent. Human-level ai's killer application: Interactive computer games. In H. A. Kautz and B. W. Porter, editors, *AAAI/IAAI*, pages 1171–1178. AAAI Press / The MIT Press, 2000.

[24] A. Micić, D. Arnarsson, and V. Jónsson. Developing Game AI for the Real-Time Strategy Game StarCraft. Technical report, Reykjavik University, 2011.

[25] I. Millington and J. Funge. *Artificial Intelligence for Games*, chapter 5 Decision Making, pages 401–425. Morgan Kaufmann, second edition edition, 2009.

[26] J. M. Olsen. Attractors and repulsors. In *Game Programming Gems 4*. Charles River Media, Inc., Rockland, MA, USA, 2002.

[27] S. J. Partington and J. J. Bryson. The behavior oriented design of an unreal tournament character. In *Intelligent Virtual Agents*, pages 466–477, 2005.

[28] T. Sandberg and J. Togelius. Evolutionary Multi-Agent potential field based AI approach for SSC scenarios in RTS games. Master's thesis, IT University Copenhagen, February 2011.

[29] T. L. Taylor. *Raising the Stakes: E-Sports and the Professionalization of Computer Gaming*. The MIT Press, 2012.

[30] C. D. Ward and P. I. Cowling. Monte carlo search applied to card selection in magic: The gathering. In P. L. Lanzi, editor, *CIG*, pages 9–16. IEEE, 2009.

[31] B. Weber, M. Mateas, and A. Jhala. Applying Goal-Driven autonomy to StarCraft. In *Proceedings of the Sixth Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2010.

[32] B. Weber, P. Mawhorter, M. Mateas, and A. Jhala. Reactive planning idioms for multi-scale game ai. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 115–122, 2010.