Agile Behaviour Design: A Design Approach for Structuring Game Characters and Interactions

Swen E. Gaudl

Falmouth University, MetaMakers Institute swen.gaudl@gmail.com

Abstract. In this paper, a novel design methodology—Agile Behaviour Design—is presented which accommodates the requirements for developing complex game agents suitable from industrial environments. An essential part of the design approach is to supported independent work of both designers and programmers by reducing bottleneck situations. The approach then fosters the creation of more loose and fluid interactions between design and implementation leaving more freedom for creative expression.

Keywords: agent design, authoring tools, planning, iva

1 Introduction

In game development and similarly in other dynamic software projects, scrum is the dominating approach [13, 9] for developing products in a managed way. The scrum process model is based on the agile philosophy allowing alterations and new features to be introduced late in the project, a situation commonly required in game development. "scrum for games" [9] an approach specifically adjusted for games discusses four phases which partition the development, i.e. concept, pre-production, production, post-production. The phases have defined milestone points that are generated at the start of the project. Even for non-industrial game development or game AI design, process models are useful as they provide a common framework for integrating different parties into a shared project that converges on a common goal. However, "scrum for games" is described only on a high abstraction level not aiding the design of specific game components such as the game AI system.

In contrast to scrum, the bod methodology [2] describes an iterative process working in similar ways to agile processes that does not focus on converging against a more stable or final product. The steps to arrive at the desired outcome in BOD are not designed for developing larger AI systems as it was originally not created with a focus on *in-time* development and the design of large collections of behaviours, but as an approach to model natural intelligence or more specifically action selection in a more accessible way. Thus, to leverage the accessibility and ease of use of BOD while at the same time making the process more controllable, design concepts from [9] are integrated into a new approach to enhance the process while making it similar to the one already used in game development.

Traditional game development phases influence and impact the creative process and freedom of designers and influence the system design. The more mature the system or game becomes, the more restrictive are deviations from the initial design. Thus, features need to be known early and should not emerge late in the design. This affects character and interaction design massively as alterations to them emerge during testing. Here, AGILE BEHAVIOUR DESIGN can support the process and the development of behaviour-based agent AI.

2 Background & Related Work

The original approach to BOD is a top-down analysis of a desired behaviour combined with a bottom-up generation of plans and the behaviour library[1]. The top-down analysis starts with the definition of a high-level task the agent wants to achieve, an undertaking for generating a single agent in a well-defined environment by an expert. Next the plans are build bottom-up by implementing primitives and enriching the behaviour plan. Primitives are clustered into behaviours according to their usage of shared memory/state objects. New goals and sub-trees are added until the agent is capable of performing the initially defined task. An existing systems analysis, comparing three INTELLIGENT VIRTUAL AGENT (IVA) approaches—BOD[2], ABL[10] & FATIMA [4] , found that the decomposition and plan creation to be a challenging process [7]. Thus, supporting this step is crucial for reducing the burden on novice or less technical users. Similar observations were made during an under-graduate AI course taught at the University of Bath, where student as part of their coursework created IVAs using BOD. Novices tended to generate either flat shallow plans or deep narrow plans, restricting the resulting agents immensely. These observations also apply to other approaches such as BEHAVIORTREE (BT) [3].

When using BOD, iterating over the plan and creating new behaviour primitives does result in a tight coupling of programmer and designer as the complete behaviour library and plan structure is in flux. This is undesirable as it locks both parties into restrictive patterns of interaction.

Based on the BOD decomposition methodology [2], a minimal plan is created that starts out as a boiled down version showing a proof of concept implementation [12]. In their application to UNREAL TOURNAMENT, Partington and Bryson describe the initial agent task as to capture the enemy flag from the opponent base [12]. After the decomposition, a list of action sequences exists and a plan which contains limited functionality. The resulting agent is able to perform a basic task such a moving in a direction but not sufficing the original goal.

From this initial prototype, the plan incrementally becomes more complex by adding new elements. While increasing the complexity, the first decomposition and primitives list get adjusted as new behaviour needs to be integrated. However, this process requires revisiting the underlying behaviour library, a process which creates forces a tight communication of designer and programmer. The

[1] Behaviour plans are designed to be human readable/amendable. The behaviour library is compiled game engine or agent framework specific program code.

process also requires altering the intended high-level task, leading many times to re-visiting the existing plan.

With increased agent sizes, the complexity of the underlying behaviour library and plan structure grows as well. So selecting the best system architectures for an approach is important. If the system is based on finite-state machines, the complexity would, in the average case, increase exponentially which would render any system at a certain complexity unusable. With approaches such as BT and POSH [6, 5], the complexity grows only in the worst case exponentially. Another approach are planning systems such as GOAP [11], they require expert knowledge of the plan to predict the outcome but reduce the interdependence of nodes and the amount of manual checking transitions. POSH integrates a lightweight planner allowing local design by modifying existing sub-trees and hierarchically nesting them within its modular structure.

3 A Directed Model to Behaviour Design

To advance the BOD approach it is possible to consider parts of the SCRUM process when designing an agent. Scrum is an agile software development process integrating iterative development and testing while maintaining as much as possible the time predictability from other development processes such as the Waterfall model. It partitions the project into smaller *Sprints* each taking a specified time and dealing with a defined set of features/tasks. At the end of each *Sprint*, the entire system should be able to execute the features developed during the Sprint, including those that have been newly integrated. Features are collected on a feature board which presents them in ordered lists (product backlog) of completed, in-progress and to be implemented elements. "Scrum for Games" [9] starts with an initial full specification of the system and continuous stable versions of the product while incrementally adding features from a feature board. The important part is the feature board; it is created and laid out to schedule the work and progress of all features. This contrasts traditional SCRUM where work is scheduled into tasks that potentially are multiple or partial features. The work on the product starts after all features for the final product and production phases have been laid out. This process has similarities with BOD is similar but as shown by [12], the starting point is a minimal plan and thus a minimal set of action primitives.

Agile Behaviour Design

The first step is to decompose a given scenario into a full set of behaviour primitives(i.e. actions and senses) and state variables. After that, the designer is building a full behaviour plan for the agent that suffices the scenario specification. This step is more time-consuming then the incremental build up using BOD and cognitively more challenging. It should be done in as few sessions as possible building an entire behaviour plan bottom up using the previously specified primitives. This part of the development is a pure design stage without the need for programmer involvement.

Next, the initial design plan is evaluated together with a programmer. Based on the feedback, the design is modified; primitives are added, adjusted and renamed. Behaviour stubs are generated in a OOD fashion, all specified primitive are stubbed and clustered into them according to memory/state usage. This stage is a pure programmer task. New action primitives should contain a default return state. At this point, the fallback action which should be called if other plan elements fail is the only action which needs an implementation. This action allows the plan to be executable and represents an idle state of the agent.

When designing a behaviour plan its sub-plans (sub-trees in BT) are triggered upon meeting a condition. Thus, when the conditions are not fulfilled for a single sense the trigger does not release the related sub-tree. Using this mechanism, it is possible to deactivate parts of the plan similar to the bitmasks used by Isla [8]. To achieve this, the designer can integrate senses that unlock sub-trees if they are triggered, thus once implemented they can activate the sub-plan.

After obtaining a first feature-complete plan, the work on the underlying behaviour primitives can be adjusted to work on individual features. Thus, the feature board can be ordered by clustering actions and senses under specific features. The alteration to the feature board can be done by grouping actions and senses according to their position in the hierarchical tree. This supports the identification of redundant or re-usable functionality by identifying similar usage of actions and senses within competences.

On the feature board, the relating features should be ordered so that sub-trees of the plan can be completed one at a time and thus unlocking them for the agent. This clustering of features allows programmers to shift entire feature blocks up and down on the feature board without impacting other sub-trees.

If the behaviour designer now decides to alter the plan, a large number of actions and senses are already stubbed within the hollow behaviour set. This given structure allows the designer to work independently on the design while programmers can implement the stubs. Following this approach requires fewer inclusions of new underlying primitives than following a simple incremental approach; it also distributes the work better between designer and programmer by initially close coordination in the first phase and a looser coupling later on.

Ideally, the work is directed from bottom to top of plan following the idea of the SUBSUMPTION design of [1]. This will enable higher level drives after lower level ones have been implemented and tested. By approaching the design this way it increases the complexity of the agent according to the designed priorities without impacting the robustness or completeness of the behaviour plan.

Agile Process Steps

1 Decompose scenario behaviour into primitives, states and goals (Design)
2 Design full behaviour plan that would suffice intended scenario (Design)
3 adjust/alter plan and primitive list (Design+Programming)
4a Templating behaviour stubs (Programming)
4b Design behaviour plan to have feature locks (Design)
5 Modify feature board & sort according to sub-trees (Design+Programming)

6a Implement stubs and alter primitives according to features (Programming)
6b Test & Develop behaviour plan based on given primitives/features (Design)
7 Loop to 3) until feature board is empty

Agile Behaviour Design was used to develop agents for starcraft [5] and was utilised in the development of the Android game Stealthier POSH[2] as a prove on concept. Maintaining a prioritised feature set which relates to the sub-trees proved in those two case studies beneficial and bridges the process to commercial games. The feature board allows for better tracking the development progress and more independent work of designers and programmers. Additionally, it removes the burden of numerous changes to the behaviour library early in the project or restricting the designer from working purely on the plan without being able to test it. Unimplemented actions returning the default state allow for the parallel work on partial behaviours which decouples the programmer.

Using hierarchical planners such as posh in combination with Agile Behaviour Design it is possible to work on smaller sections of an agent concentrating for example on interaction with other agents while the dependencies between designer and programmer are reduced. Additionally integrating the default trigger states, sub-trees unlock based on the progress of their underlying implementation. This cascaded unlocking of the tree and the resulting behaviour allows for a better version control of the behaviour library because it is more directed towards realising connected sub-trees. The combination of working on sub-trees and the feature board based on scrum directs the agent implementation to focus on connected pieces. The new approach should provide sufficient support for working on more complex systems or distributing work between different people such as movement and narration design for a given agent.

4 Conclusion & Future Work

This paper presents a novel, project-oriented alteration to the existing behavior oriented design (bod) [2]. The focus of the new methodology is to provide better separation of design and programming and to support the development of artificial agents in teams of multi-disciplinary authors. The two case studies and the feedback from the systems analysis [7] create the basis for the newly introduced process steps of the methodology. This new process allows designers and programmers to distribute their work better while still following keeping the project progress in mind. Agile Behaviour Design reduces the dependencies of the different user groups. To further aid the development and to focus on multi-platform development a new arbitration architecture was proposed—posh-sharp [5]—which extends Bryson's original concept of posh to support the agile design approach better. As a next step, further evaluations of the new methodology and approach are intended with novice and expert users as well as a widened systematic analysis of development approaches to support cross-disciplinary design.

[2] The game is available on the Android app store or using the following link: `https://play.google.com/store/apps/details?id=com.fairrats.POSH`

References

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

[2] Bryson, J.J.: Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents. Ph.D. thesis, MIT, Department of EECS, Cambridge, MA (June 2001), Tech Report 2001-003

[3] Champandard, A.J., Dunstan, P.: The behavior tree starter kit. In: Rabin, S. (ed.) Game AI Pro: Collected Wisdom of Game AI Professionals, pp. 72–92. Game Ai Pro, A. K. Peters, Ltd. (2013)

[4] Dias, J., Mascarenhas, S., Paiva, A.: Fatima modular: Towards an agent architecture with a generic appraisal framework. In: Emotion Modeling, pp. 44–56. Springer (2014)

[5] Gaudl, S.E.: Building Robust Real-Time Game AI: Simplifying & Automating Integral Process Steps in Multi-Platform Design. Ph.D. thesis, Department of Computer Science, University of Bath (2016)

[6] Gaudl, S.E., Davies, S., Bryson, J.J.: Behaviour oriented design for real-time-strategy games – an approach on iterative development for starcraft ai. In: Proceedings of the Foundations of Digital Games. pp. 198–205. Society for the Advancement of Science of Digital Games (2013)

[7] Grow, A., Gaudl, S.E., Gomes, P.F., Mateas, M., Wardrip-Fruin, N.: A methodology for requirements analysis of ai architecture authoring tools. In: Foundations of Digital Games 2014. Society for the Advancement of the Science of Digital Games (2014)

[8] Isla, D.: GDC 2005 proceeding: Handling complexity in the halo 2 AI. `http://www.gamasutra.com/view/feature/2250/gdc_2005_proceeding_handling` (2005), [Accessed 4th Apr 2017]

[9] Keith, C.: Agile Game Development with Scrum. Addison-Wesley Signature Series (Cohn), Pearson Education (2010),

[10] Mateas, M., Stern, A.: A behavior language for story-based believable agents. Intelligent Systems, IEEE 17(4), 39–47 (2002)

[11] Orkin, J.: Agent architecture considerations for real-time planning in games. In: Young, M.R., John, L. (eds.) Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference. pp. 105–110. AAAI Press, Menlo Park, CA (2005)

[12] Partington, S.J., Bryson, J.J.: The Behavior Oriented Design of an Unreal Tournament character. In: Panayiotopoulos, T., Gratch, J., Aylett, R., Ballin, D., Olivier, P., Rist, T. (eds.) The Fifth International Working Conference on Intelligent Virtual Agents. pp. 466–477. Springer, Kos, Greece (September 2005)

[13] Rubin, K.: Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley Signature Series (Cohn), Pearson Education (2012),