Literature Review

Joshua Pritchard

University of Huddersfield

N421: BSc (Hons) Computing Science with Games Programming SW

CHP2524: Final Year Project

Dr. Minsi Chen

December 14 2020

Literature Review

Game Development Difficulty

There is a general consensus that developing video games has a high level of difficulty. This theme is well-established, with a crucial article by Blow (2007) explaining how increasing consumer expectations, software framework sizes and lack of external tool support have brought on said difficulty. Despite the area of games development being one in which research quickly gains obsolescence, and an influx of the accessible software discussed with, the key issues raised still remain. Almost every piece of games development literature references difficulty in some form, with multiple studies emphasizing creating behaviour for games (Pellens et al., 2008; Roedavan et al., 2020; Kounoukla et al., 2016). Any usage of the design patterns detailed by Kounoukla et al. (2016) comes with significant implementation difficulty (Budinsky et al., 1996), and leads to a contribution towards the statistic of just 16% of projects being completed on time and under-budget (Kanode & Haddad, 2009). Various studies highlight the extenuating considerations and skillsets required in game development (Aleem et al. 2016; Sarinho & Apolinário, 2009), while another key study uses a similar analysis to highlight the need for a streamlining of the entire process (Cutumisu, et al., 2007). Despite this, there is a lack of research and standardization on key elements of games development, such as the creation of gameplay mechanics.

Learning Gameplay Programming

The discussed general difficulty of games development, and the author's own experiences present a more granular issue; It is hard to learn gameplay programming *specifically*. Few studies deal with this exact topic, however Roedavan et al. (2020) details how 33% of students reported difficulty in creating gameplay with Unity C#, and that the majority of students find difficulty in implementing *mechanic* logic within *code* logic. This is reinforced by Kounoukla et al. (2016), who explain that despite 'Game Design Patterns' being well documented (thanks in large part to the works of Björk (n.d.)), there is little to no guidance on their implementation in code. They further their argument for design patterns by detailing

how common GoF patterns can be used for common game mechanics, along with their benefits. The topic is indirectly supported via lack of mention by Kanode & Haddad (2009), where much is written of iteration to 'find the fun' of a game, but nothing is written about simply the difficulties of gameplay programming from a software engineering perspective. A potentially crucial explanation for this gap in knowledge comes from Aleem et al. (2016), drawing attention to the fact that researchers don't have the resources to create large games, and developers never publish the results of their own efforts. While the knowledge gap is clearly receding, there is still a critical lack of feedback loops between games development literature and practice.

Connections Between Academia and Industry

The prior suggestions lead to an obvious theme – There is a mismatch between the works of academia and industry in games development. Aleem et al. (2016) puts the point in context succinctly, explaining that "game developers prioritize the game development process by rapid creation and implementation of content", whereas "scientists and researchers prioritize investigation and research into the individual components of a system". A recent study, its criticisms hold well today, however are furthered by relatively few. Kounoukla et al. (2016) indirectly support this notion through their concluding of actual games not (or accidentally) using GoF design patterns. Said study was limited in that only Open Source Software games could be analysed, however this author sees that as a criticism of the games industry, not the study. Maranh et al. (2017) put forth a potential criticism of academic games mechanic representations, in that they are complex constructions and formally defined. Despite the criticism of *both* factions in literature, little has been done to expand upon and/or suggest solutions to the (growing) problem.

Existing Mechanic Definitions

Already touched upon with Maranh et al. (2017), the mismatch theme brings about a more pressing issue for the granular topic of games mechanics; They are defined either too formally, or applied

too simplistically in academia to represent any benefit to practitioners. Multiple studies touch on AI mechanic generation (a separate theme in itself), however the work of Zook & Riedl (2014) best showcases the simplicity with which academia views game mechanics with the simplistic game environments that their mechanic generator is applied to. The observations of Maranh et al. (2017) have already been discussed, however Thompson, et al. (2013) also support the notion of over-formality in their foundation criticisms of existing game description languages being unable to handle even simple arcade games. The date of this work adds further criticism to formal representations with the dates of the titles that they claim GDLs could not support: *Space Invaders (1978), Asteroids (1979)*. However, despite the criticism, both works made significant strides in rectifying the issue, with Thompson, et al. (2013) developing the lacked game description language, and Maranh et al. (2017) making a key observation around players viewing mechanics as abstracted and simplified forms of game rules, leading to their well-structured and seemingly obvious methodology for game mechanics analysis and definition. Despite said key strides taken by the two authors discussed, it is clearly an under-pursued area in literature, with large gaps between incremental research (to date, Maranh et al. (2017) have but 3 citations).

Games Mechanics Development Literature

Given the preferences established for both researchers and industry in games development, the low level, formal focus is somewhat unsurprising. What *is* surprising however, is how there seems to be a general lack of *any* literature covering the creation/implementation/standardization of high level games mechanics. Maranh et al. (2017) offers justification for how this state has arisen, with an observation pointing towards a lack of any agreed upon definition for game mechanics between authors. 3 years prior, Guana & Stroulia (2014) display a similar thought towards how multiple games prototyping tools *exist*, yet academic articles on their technical implementations and ease of use are lacking. Some studies reference again specifically how Game Design Patterns (Björk, n.d.) come with no implementation guidance at all (Kounoukla et al., 2016; Pellens et al., 2008). Furthermore, relatively critical studies in specifically games

development fail to mention *anything* towards the issue, even in passing (Blow, 2007; Kanode & Haddad, 2009). Aleem et al. (2016) support the notion, explaining how game development reflects only general state of the art practices in software engineering, and that key factors within the game development process are the *least* addressed area in software games research. However, they also make a key observation in that the management of games development has become harder than anyone thought it would, and because of its fragmented nature, no clear direction for advancement can be found in literature. This final, crucial observation both explains the gap in literature and points another finger towards the game development industry where knowledge sharing is concerned.

Game Development Tool Effectiveness

Running parallel to the last point, there runs a consistent theme throughout the years of whatever current games development tools in existence at the time being inadequate. The key study in this area as relevant to this project is the work of Cutumisu et al. (2007). In writing, they highlight that despite tools being made for gameplay scripting, they [*Kismet*] were too low level and hard to use. They further state that the manual scripting approach Kismet was built for does not scale up to large games development, and that there was an "urgent need for tools to simplify the game content creation process, including specifying, implementing, and testing game content". 13 years later, this study has its limitations, most notably Kismet not existing anymore, replaced by *Blueprints*, for the same purpose. Despite this, the approach to manual scripting described has not changed, and while the need may be less urgent, it is still present in industry today, especially where the solution of generative/adaptive tools are relevant. A variety of studies written around the same time period support this point (Blow, 2007; Pellens et al., 2008; Pellens et al., 2009), with the latter making a crucially relevant observation that there are no tools that focus on the creation of gameplay or game story. By 2014, tools were available, however there was still a discussed need of environments to allow *non* computer experts to perform the prototyping process (Guana & Stroulia, 2014). Overall, the state of games development tools has improved dramatically over

the course of relevant literatures. Despite this, the key point of Cutumisu et al. (2007) in their discussions of generative/adaptive process to solve the scaling-up issue goes relatively unnoticed and all but forgotten about.

Adaptive Programming for Games

That being said, a small number of studies *have* worked to apply generative/adaptive programming in games development. Once again, the key study as relevant to this project is the work of Cutumisu et al. (2007), who detail the application of *ScriptEase* to the cRPG *Neverwinter Nights*. ScriptEase applies the adaptive programming paradigm to game scripting by employing a 3 step process in which authors first select a high level story pattern, make small adaptations to it, and then click a button to generate the code necessary for that pattern to occur in the game world. The authors use a case study on non-programmers to justify their claims that adaptive programming is the future of programming, and that is has the potential to affect users in various domains, any take steps towards eliminating programmers from many content authoring processes. Within, they make two crucial points. One, a comparison to constructive programming that states asking authors to create new scripts every time is akin to asking them to create new *patterns* every time they were to use ScriptEase. The second, an evidenced theory that adaptive programming allows non-programmers to create complex programs. As already discussed, the date of this work is a limiting factor in the external observations and criticisms they make, however the novel aspect they identify is one of the future and in many ways *more* relevant now than it was in 2007. Since then, just two studies seem to have applied even similar concepts to those discussed. Firstly, Zook & Riedl (2014) who touch on an application of AI mechanic generation in adaptation, wherein mechanics could be iterated upon to achieve adaptation requirements (in light of new content design requirements). The second is Guana & Stroulia (2014) who define a code generation environment for physics based games, using a text editor backed by a domain specific language. This allows for adaptations to be made to game actors in terms of finite, predefined sets of properties in order

to adapt gameplay at a high level. In this adaptive programming paradigm, the last article discussed is the closest literature has come to documenting an application of adaptive programming in gameplay, making this an area of research ripe for innovation, 13 years after the inspiring ScriptEase.

Existing Applications of Adaptive Programming to Games Mechanics

The prior discussions lead to an interesting theme, that adaptive programming is a proven concept, yet it has not been applied to gameplay mechanics. This topic appears to be indirectly supported by a number of academic articles. Cutumisu et al. (2007) make reference to their methods being used in 'other' domains, but their work rests around applying it to game story authoring. Blow (2007) provides a summary of then current tool developments, none of which apply adaptive programming. Thompson, et al. (2013) identify ongoing work to realise the potential of their description language in the areas of procedural content generation and automatic game design, however the lack of product and/or citations of articles such as Cutumisu et al. (2007) and Maranh et al. (2017) show that as of yet, there still remains an opening in literature for an application of adaptive programming to specifically games mechanics.

Existing Applications of Adaptive Programming to Other Domains

This author's experiences point towards a justifying theme for said lack of application in that the majority of research tied to adaptive programming gets applied to AI or the construction of entire games. This theme is entrenched in existing literature, with numerous studies applying their work to artificial intelligence scripting/generators (Zook & Riedl, 2014; Pellens et al., 2008), others applying it towards entire game generation (Sarinho & Apolinário, 2009), and one even applying to entire games from the *basis* of general video game playing research (Thompson, et al., 2013). This theme was already in motion in 2007 (Cutumisu, et al., 2007), and despite the acclaim many of these articles have in their advancements, it is a relatively recent criticism that states many existing authoring environments are designed to meet the production of a full video game (Guana & Stroulia, 2014). It is through said design that the authors explain these tools are unsuitable for quick game prototyping and testing. This theme

serves to enhance the idea that there is a gap for the application of generative/adaptive programming to domains other than AI and entire games (such as gameplay mechanics).

Applying the Adaptive Programming Paradigm to Games Mechanics

What *this* project is proposing is an application of the adaptive programming paradigm to the creation of individual gameplay mechanics. This is an idea supported and justified by many, however never referred to explicitly by name. The most directly related of these directly suggests benefit to inclusive programming interfaces and the creation of mechanic stereotype catalogues & manuals with best practices (Maranh et al., 2017). Two studies dealing with granular areas of game behaviour generation detail their workings and benefits, with the latter suggesting that distinct code generation environments can made for different game genres (Pellens et al., 2008; Guana & Stroulia, 2014). Multiple studies re-affirm the prototyping, code re-use and software quality benefits of design patterns (both generative and descriptive), with the latter adding a second advocation for sample implementations of game mechanics that can be re-used as starting points for source code development (MacDonald, et al., 2002; Pellens et al., 2009; Kounoukla et al., 2016). Aleem et al. (2016) provide a solid basis for justification of work with their observations around novel games being produced by simply iterating over gameplays etc. and that game development must follow an incremental model. Finally, Cutumisu et al. (2007) provide an idyllic future outlook where an author can specify and test games directly, without ever using a programmer. Clearly, any lofty ambitions are far from realisation, however the desire and justification for such an application have been growing for over a decade.

Making a Tool to Generate Gameplay Mechanics

It is evident then, from the prior contents of this literature review, that the games development community has a software tool shaped gap in the area of gameplay programming. The crucial article by Blow (2007) ultimately calls for better tools and workflow in order to create better games, something that the usage of Zetcil (Roedavan et al., 2020) by students to prototype games leading to a 36% increase in

their confidence re-affirms 13 years later. Two other documented tools using the generative design patterns basis of MacDonald, et al. (2002) further justify the generative capabilities of tools in the game development domain (Cutumisu, et al., 2007; Guana & Stroulia, 2014), with the latter heavily citing software architecture as a benefit of generation, something that relatively recent games mechanics studies claim would be benefited by their respective specifications and design pattern mappings (Maranh et al., 2017; Kounoukla et al., 2016). The ambitious applications of the works by Thompson, et al. (2013), Pellens et al. (2009) and Sarinho & Apolinário (2009), born in part from Budinsky et al. (1996)'s statement that users find the manual implementation of design patterns either too difficult or a chore, are living justifications of the same author's profound claim that application is the necessary first step to profitable automation. What other justification is required, aside from the quality, usability, applicability and conclusion that success is *linked* to quality (Aleem et al., 2016), for it to be evident that now is the time for a tool of this nature to be created?

Benefit to Students and Prototypers

Following from the last point, it is apparent that such a tool would benefit both games prototypers and students. The most vocal of support for this comes from Guana & Stroulia (2014), who explain that the benefits of code generation environments include closing the gap between designers and programmers, with the former able to realise a vision with high level semantics, and the latter able to become part of the pre-production process by making small adjustments to code that cannot be generated. Through evaluation of their tool they found a 96% reduction in prototype creation time, and that 80-100% of code could be generated automatically for high school students using the tool, claims corroborated by Roedavan et al. (2020). Multiple studies promote the benefits of the design patterns that the generated code could use (Kounoukla et al., 2016; Barakat, 2019), however MacDonald et al. (2002) made the point over a decade before that novice programmers (students) have difficulty implementing them, whereas experienced programmers (prototypers) find it a chore. It is evident then that this tool, a

bridge between required collaboration between research and developers, would instill desired software architecture best practices from the beginning of both education and project conception Aleem et al. (2016).

Bibliography

Aleem, S., Capretz, L. F., & Ahmed, F. (2016). Critical Success Factors to Improve the Game Development Process from a Developer's Perspective. *Journal of Computer Science and Technology*, 925-950.

Barakat, N. (2019). A Framework for integrating software design patterns with game design framework. *Proceedings of the 2019 8th international conference on Software and Information Engineering*.

Björk, S. (n.d.). *Main Page*. Game Design Patterns: http://virt10.itu.chalmers.se/index.php/Main_Page

Blow, J. (2007). Game Development: Harder Than You Think. *Association for Computing Machinery*, 28-37.

Budinsky, F. J., Finnie, M. A., Vlissides, J., & Yu, P. S. (1996). Automatic Code Generation from Design Patterns. *IBM Syst. J.*, 151-171.

Cutumisu, M., Onuczko, C., McNaughton, M., Roy, T., Schaeffer, J., Schumacher, A., . . . Gillis, S. (2007). ScriptEase: A generative/adaptive programming paradigm for game scripting. *Science of Computer Programming*, 32-58.

Guana, V., & Stroulia, E. (2014). PhyDSL: A Code-generation Environment for 2D Physics-based Games.

Kanode, C. M., & Haddad, H. M. (2009). Software Engineering Challenges in Game Development. *2009 Sixth International Conference on Information Technology: New Generations*, 260-265.

Kounoukla, X.-C., Ampatzoglou, A., & Anagnostopoulos, K. (2016). *Implementing Game Mechanics with GoF Design Patterns.*

MacDonald, S., Szafron, D., Schaeffer, J., Anvik, J., Bromling, S., & Tan, K. (2002). Generative design patterns. *Proceedings 17th IEEE International Conference on Automated Software Engineering*, 23-34.

Maranh, D. M., de Oliveria da Rocha Franco, A., & G. R. Maia, J. (2017). Towards a Comprehensive Model for Analysis and Definition of Game Mechanics.

Pellens, B., De Troyer, O., & Kleinermann, F. (2008). CoDePA: a conceptual design pattern approach to model behavior for X3D worlds. *Proceedings of the 13th international symposium on 3D web technology* (pp. 91-99). Los Angeles: Association for Computing Machinery.

Pellens, B., Troyer, O., & Kleinermann, F. (2009). Visual generative behavior patterns to facilitate game development.

Roedavan, R., Agus, P., Korio Utoro, R., & Putri Sujana, A. (2020). Zetcil: Game Mechanic Framework for Unity Game Engine. *International Journal on Artificial Intelligence Tools*, 96-105.

Sarinho, V. T., & Apolinário, A. L. (2009). A Generative Programming Approach for Game Development. *VIII Brazilian Symposium on Games and Digital Entertainment* (pp. 83-92). Rio de Janeiro: IEEE.

Thompson, T., Ebner, M., Schaul, T., Levine, J., Lucas, S., & Togelius, J. (2013). Towards a Video Game Description Language. *Dagstuhl Follow-ups*, 85.

Zook, A., & Riedl, M. O. (2014). Generating and Adapting Game Mechanics.