Scalable Level Generation for 2D Platforming Games

Neall Dewsbury[1], Aimie Nunn[2], Matthew Syrett[*3], James Tatum[2], and Tommy Thompson[†3]
[1]University of Derby, Derby, UK
[2]Table Flip Games Ltd, UK
[3]Anglia Ruskin University, Cambridge, UK

ABSTRACT

In this paper we present a model for procedural generation of 2D platforming levels, with the aim to ensure content can be scaled as players progress. Levels are generated through use of a two-phased generate and test approach, with the first reliant upon a grammar for generation of activities, while the latter is focussed on the positioning of geometry. These methods are made scalable courtesy of a budget-driven approach that limits the expressiveness of each component. We investigate the effectiveness of this approach and the playable levels it can generate for a 2D 'infinite runner' video game.

Keywords

procedural content generation, games, levels, 2D platformers

INTRODUCTION

In recent years procedural content generation (PCG) - the practice of asset and content creation through algorithmic processes - has found new purchase as an academic pursuit for artificial intelligence algorithms. This new-found academic influence has placed emphasis on the quality of content generated as well as the flexibility of the systems prescribed. Early PCG research focussed on building systems that could create large amounts of content but without real consideration for whether this content would prove of value or interest to a player as their overall skills matured over time.

The issue of quality assurance for a generative system is paramount, given that generative systems left to create content unsupervised may result in products that are incongruous to the setting. This is even more pertinent when generated content is critical to the structure of the game and may result in designers expressing specific constraints on the system. This can be seen when considering the likes of levels in *Spelunky* (Mossmouth, 2009) in contrast with 'item drops' in games such as *Borderlands 2* (Gearbox, 2012) or level textures in *Tiny Wings* (Andreas Illiger, 2011). Should a procedurally generated item or level texture prove discordant with players expectations, they can decide to ignore them provided the game maintains the promise of 'better' content in the future[1]. However, dysfunctional or discordant levels may prove detrimental to a players overall gameplay experience, given that it could inhibit a player's progression and their enjoyment along with it.

[*]matthew.syrett@anglia.ac.uk
[†]tommy@t2thompson.com

[1]We adopt the term 'better' as representative of the subjective opinions of a typical player, rather than any consensus of the authors derived from established metrics.

Proceedings of 1st International Joint Conference of DiGRA and FDG

Quality assurance is not a solved problem by any means, but efforts have been taken to address this through a variety of methods: assessing the levels of interaction between players and generated content [8], extrapolating rules of human design to be used as part of the generative process [4] and crafting autonomous critics based upon expected player performance [11]. While this research is largely focussed on establishing player-driven and/or modelled metrics, the focus of this paper is interested in managing these issues - initially - from the generative system itself: managing not only the need to ensure a range of functional, high-quality content but to also address the notions of *variability* and *scale*. This allows not only for variety in the generative system, but content that can subsequently increase in difficulty when placed within a games defined rules. This is particularly relevant when dealing with the creation of gameplay spaces such as platformer or adventure games: where we may wish for levels, dungeons, puzzles or monsters to be varied in their design but gradually increase in challenge the longer we play - all the while ensuring they are functionally sound. By managing scale - and in-turn challenge - in the generator, we can aim to provide a sense of progression to the player while continuing to address the issues of variety and functionality as expected.

To address these concerns, we focussed on the creation of a generative system for an *infinite runner*: a sub-genre of two-dimensional platforming games. Infinite runners, as the name implies, require a generative system to continually make new content as the player progresses. This content must also become increasingly challenging for players over time and thus proves an excellent domain to initially explore our research interests. This paper explores the creation of content for 2D platformers that can be constrained by designers but without significantly impacting variability of the generated product. This is achieved courtesy of adopting existing theories in level-generation: finding inspiration in research for platformer-style games by Dahlskog & Togelius detailed in [4, 5] alongside work by Smith and Whitehead in [19]. We adopt the grammar-driven generative approach of these works and de-couple a given levels design or intent from its spatial construction: modelling the the actions we wish for players to complete independent of how those same actions manifest within the game world. The contributions of this paper are found within this multi-phase approach, as it addresses our concerns regarding variability and scale by managing the total resource or 'budget' each component of the generative system has available to it. Thus influencing the outcome of each phase of the generation process.

We begin by providing a general overview of the 'infinite runner' genre in an effort to familiarise readers with our problem space, followed by a short overview of relevant literature in the areas of level generation and more specifically platforming levels. We provide an in-depth description of the framework established that yields our scalable generative system as well as some of the individual generators that are used within the system. This is followed with an extensive analysis of the expressivity of the system under specific constraints and parameter ranges and a discussion of the types of levels this system can produce.

INFINITE RUNNERS

Infinite runners are a sub-genre of platformer video games that adopt many core principles while also introducing constraints both in terms of level design as well as gameplay mechanics. Platforming games, as typified by the likes of *Super Mario Bros.* (Nintendo EAD, 1985) and *Sonic the Hedgehog* (Sonic Team, 1991), challenge players to navigate an envi-

(a) Canabalt (Adam Saltsman, 2009)

(b) Jetpack Joyride (Halfbrick Studios, 2011)

(c) Temple Run (Imangi Studios, 2011)

(d) Spider-Man: Unlimited (Gameloft, 2014)

Figure 1: Examples of the 'infinite runner' adopting a side-scrolling (1a, 1b) or over-the-shoulder (1c, 1d) perspective in which a player must run continually until failure.

ronment comprised largely of walkways, platforms and gaps built in a tiled fashion using existing art assets. These are typically configured allowing for specific geometric features such as valleys, stairs and chasms that challenge the player.

However, infinite runners distinguish themselves from traditional platforming games in two key areas. Firstly, the number of actions and/or control axis available to players is reduced; with a specific constraint in players not being in control of forward movement. This requires players to time actions to avoid obstacles and gaps between platforms. Infinite runners typically adopt a side-scrolling or over-the-shoulder perspective in accordance with the available control axis. In the first instance (Figures 1a and 1b), players typically only need to time jump actions, whereas in the latter (Figures 1c and 1d) the emphasis is on jumping as well as left, right and down actions to dodge or slide underneath obstacles. A second key distinction of infinite runners is that levels do not typically 'end'. Instead the game world will continue to expand so long as the player is still alive: increasing in challenge and providing smaller periods of respite over time. Typically, a players final score is related to the distance they have travelled. Given that we do not know in advance how long a player will continue to progress, there is a requirement to automate this process.

For the purposes of this research, we adopt the use of *Sure Footing* (Figure 2) : a side-scrolling infinite runner game developed by the authors using the *Unity3D* engine. Players must navigate the generated terrain whilst avoiding an enemy that is chasing them from behind. Atop these platforms are either items for the player to collect or obstacles that will hinder player progress.

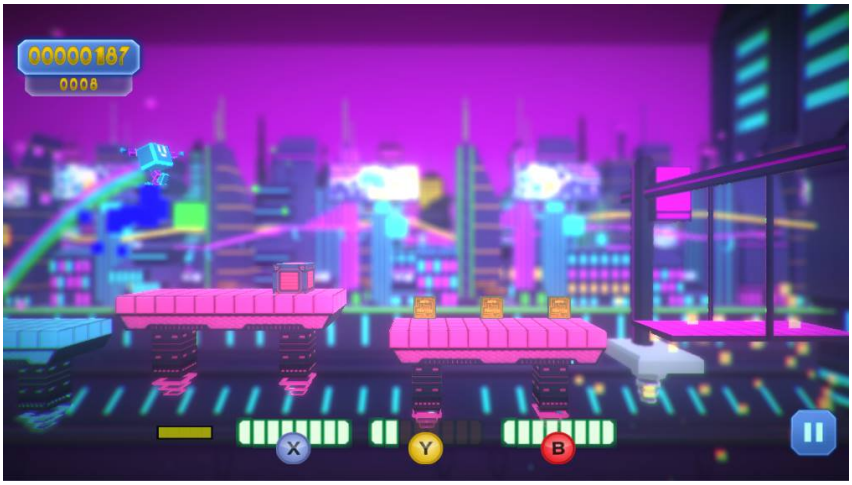


Figure 2: A screenshot of *Sure Footing*: an infinite runner game that acts as the problem domain for this research.

RELATED WORK

There is a significant body of research that exists in the area of level generation in computational intelligence (CI) and artificial intelligence (AI) research. Within this considerable amount of literature, the application of generative systems within platforming games is arguably the most prominent. As such, we present a series of works that have inspired this current phase of research, alongside some more prominent publications in the field. This is aimed at establishing not only the inspirations for this research, but also to establish where this work sits within this field of study.

The most prominent body of research in platforming generation is associated with the game *Super Mario Bros.* This has been achieved in a variety of forms, having achieved momentum courtesy of the *Mario AI Competition* [21, 23] and the introduction of the level generation track hosted between 2010 and 2012 [16]. The range of work contributed to the competition varies between 'classical' artificial intelligence methods to more machine-learning driven computational intelligence pursuits. In each instance, the system is reliant on a series of metrics recording a players initial playthrough of a sample level. This data allows for the system to tailor its output with respect to the perception of the players ability that the metrics can afford. One area of growing interest in recent years is an emphasis on understanding how to interpret interesting or sound design decisions for the purposes of level generation. There is significant work in this area found within the Mario domain: ranging from establishing grammars representative of good tenets or patterns of design [14, 5] to more recent approaches adopting machine learning techniques to either guide generators through training data in the form of levels [20] or gameplay videos [7]. Thankfully, research of platforming games is not confined solely to the *Super Mario* domain: with work detailed in [19] focussed on the creation of adaptive, grammar-driven methods for procedural generation of levels for a generic platforming game. This work subsequently adopts user-interactions to allow for mixed-initiative design of levels.

In addition, this paper continues a body of work by one of our authors that places empha-

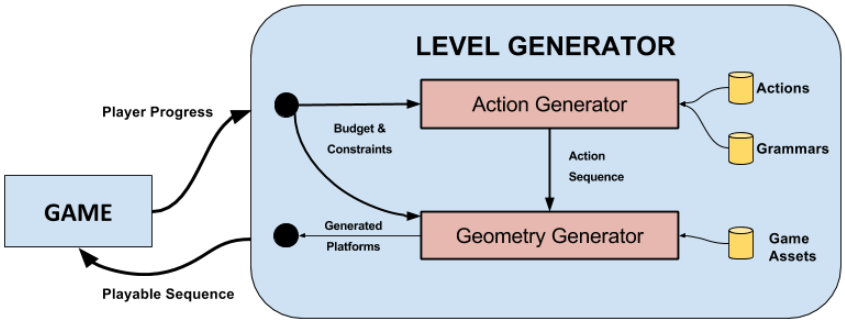


Figure 3: An overview of our level generation framework: a two-phase generate-and-test system that builds the action sequence followed by a 'physical' instantiation to be used in-game.

sis on de-coupling the context or purpose of generated content from the final product, with previous work exploring dungeon construction within *The Legend of Zelda* [10]. This generative system focussed on the construction of the gameplay space of 'Zelda' dungeons that replicate the gameplay experience expected of these interactive spaces [9]. This adopted existing research in formal grammars detailed in [6] that de-coupled the 'mission' or structure of the players experience from the physical layout of the actual dungeon. This two-phase approach permits a relative flexibility in the potential dungeons it can create even though this preliminary body of work was reliant on a limited number of room types and puzzles.

Furthermore, prior work by one author detailed in [24] addresses the encapsulation of scale in generated content. A problem domain focussed on the construction of robots to fight in a small combat domain explored the challenges of scaling resource usage against available 'budget' afforded to the system. While this prototype implementation could yield a significantly large amount of content, a budget system is introduced that limits the potential topology of robot construction. The introduction of cost metrics to each component based on its attributes also helped to limit the systems output should a designer wish to do so.

LEVEL GENERATION OVERVIEW

The level generation framework is focussed on creating a finite segment of gameplay to be placed within the game world. This system, as shown in Figure 3 is broken down into two distinct generative phases. First the *action generation* phase crafts sequence of actions for players to complete given a set of constraints and an allocated budget. This is achieved courtesy of a generative grammar approach using an alphabet and production rules crafted by a human designer. Secondly, the *geometry generation* phase translates the action sequence into assets placed within the game's virtual environment but is again governed by its own constraints and budgetary requirements.

The design of the framework is such that the inner workings of each generative system can be treated as a 'black box': allowing for a variety of permutations of actions sequences and their subsequent realisation in game space. Furthermore, specific components related to the pseudo-random nature of the generative process are shared between each of the generative systems. This ensures that the framework can be configured to reproduce desired gameplay

sequences should a designer deem this practical. This is of particular importance given the emphasis of this research on the ability to maintain an element of control over the generated artefacts whilst ensuring an element of creative freedom within each generator. Such control over generative system outputs has been adopted previously in PCG-driven video game design, with a notable example being the challenge mode found within platforming rogue-like *Spelunky* (Mossmouth, 2009).

Arguably the most important factor in this system is that each component is governed largely by the constraints set upon it. This is driven largely by the notion of a *budget*: a fixed unit of measurement that is passed as a parameter to each generator. We adopt the use of the term budget as an allusion to a system forced to work with limited resource. As discussed in the subsequent sections, how each generator operates 'within budget' is based on how it interprets budget with respect to its designated task.

ACTION GENERATOR

As shown in Figure 3, action generation is the first phase of the scaled level generation framework. The focus of action generation is to devise the action sequence of the generated level, with the work detailed in [19] in establishing a lexicon for individual segments of gameplay adopted as a starting point. However, we apply a further layer of abstraction to the action-annotation process by adopting platforming design-patterns akin to that discussed in [4]. This results in a collection of actions detailed in Table 1. The value of each action in relation to the previously discussed budget is dependant on the action generator's interpretation. Each action generator is built to act as a form of generative grammar: a formal language to encapsulate the design of levels. This is reliant upon the use of a fixed alphabet in conjunction with a finite set of production rules that transform non-terminal symbols in the alphabet into final, terminal symbols [12]. The terminal symbols expressed within the alphabet represent the actions previously defined in Table 1. For reference, we provide the terminal symbols used for these actions within the table.

The adoption of the budget system for the action generator works in relation to the generative grammar. In the event that there is still budget provided, then the execution of one or more production rules is permitted. The production rules for each grammar are tagged with a cost defined by the designer. As such, execution of production is a decision process constrained by the available remaining budget. In the event that no production rules are possible, each action generator may either adopt a local-search optimisation of the currently constructed string or run the production rules that reduce any non-terminal symbol to the empty string, and thus end the action generation process.

The action generator adopts a combination of context-free and regular grammar systems, allowing for designers to maintain a variety of interpretations of the action space such as those in Figures 4 and 5. Figure 4 is a simple random generator which enforces small constraints on which actions can follow specific gameplay segments. While Figure 5 splits the level generation into two phases: first a gain in 'intensity' by using actions that will increase player elevation, followed by a subsequent reduction in elevation. Once the sentence of the grammar has been defined by executing production rules until budget is exhausted, we then move towards translating these actions into a playable gameplay space.

Table 1: The 'action' patterns adopted for the purposes of the level generators 'action generator' component. Each action also notes the corresponding alphabet symbol used as part of the action generator grammar systems.

Action Generator Terms	
Action	**Description**
Run (r)	A flat section of terrain which the player must run across.
Ramp-Up (ru)	A slopped section of terrain with a gradual incline.
Ramp-Down (rd)	A slopped section of terrain with a gradual decline.
Stair-Up (su)	A series of short platforms closely placed that gradually increases in height.
Stair-Down (sd)	A series of short platforms closely placed that gradually decrease in height.
Spring (sp)	A segment with a non-negotiable requirement for players to land on a spring object that will launch them to a platform beyond the reach of a normal jump.
Two-Path (tp)	A temporary split in the terrain that will force players to take a higher or lower route.
HopScotch (hp)	A series of short platforms designed to force players to quickly hop between them.

The 'Safe-Random' Grammar:

$$
\begin{array}{lll}
S \rightarrow rS & S \rightarrow hpS & S \rightarrow \epsilon \\
S \rightarrow tpA & S \rightarrow sdS & A \rightarrow \epsilon \\
S \rightarrow ruA & S \rightarrow suA & \\
S \rightarrow spA & A \rightarrow rS &
\end{array}
$$

Figure 4: The production rules of the 'Safe Random' action generator grammar. This starts with the non-terminal symbol S to start the level generation, with adoption of non-terminal A to enforce specific constraints.

The 'Height Intensity' Grammar:

$$
\begin{array}{llll}
S \rightarrow GF & G \rightarrow \epsilon & F \rightarrow \epsilon & \\
G \rightarrow suG & G \rightarrow hpG & G \rightarrow spG & G \rightarrow ruG \\
F \rightarrow sdF & F \rightarrow rdF & F \rightarrow rF &
\end{array}
$$

Figure 5: The production rules of the 'Height Intensity' grammar. Starting with non-terminal symbol S, it migrates between the 'intensity gain' (G) and 'intensity fall' (F) phases.

GEOMETRY GENERATOR

Once the action generator has derived an expression of intended gameplay activity, the geometry generator is tasked with translating the actions contained within into a playable gameplay segment. What is of paramount importance is not only must each terminal symbol expressed within the action generator translate into a specific physical[2] permutation of in-game assets, it must also retain an element of flexibility in how this can be achieved.

For the purposes of this paper - and indeed retaining an element of scope to our initial research investigation - only one geometry generator is adopted for purposes of evaluation within this paper. Each generator is responsible for solving the constraint satisfaction problem of ensuring there is a permutation of the action sequence within the provided *geometry budget*. This budget limits the flexibility of the system in terms of how much resource can be used to bring each individual action into the game space.

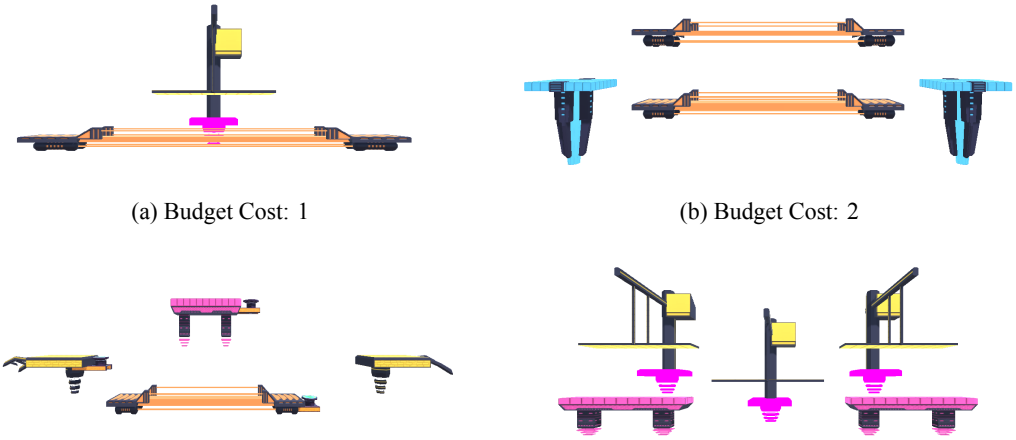
The geometry generator discussed in this paper adopts a local search optimisation approach by utilising an asset database provided by the generation framework. This database maintains a collection of platforms and other pieces of fixed geometry developed by 3D artists to be used in the game. Each asset is tagged with the possible actions it can be used to represent and the associated budgetary cost of that item. Following our previous discussion of the inspiration of design patterns for platforming games in [4, 19, 5], this approach allows for designers to craft as many unique permutations of the same design pattern as they see fit. Furthermore, the level of abstraction denoted for a given asset can vary in scale, with the use of individual platforms and geometric shapes to more complex collections handcrafted to represent specific activities much akin to the works detailed in [4]. This can be seen in Figure 6 as a collection of pre-built platforming sequences all aimed to satisfy the same core 'two-path' design pattern: allowing for two paths for players to navigate within a short gameplay segment. Each permutation has it's own budgetary cost dictated by the perceived difficulty. As such, given that 'cheaper' platforming segments are easier to navigate, an increased budget being provided to the same action sequence will yield more complex levels.

The subsequent placement of these platforms is reliant upon an understanding of finding the 'connecting' points and establishing a small gap between them for the player to jump across. While not pertinent to the focus of this paper, it is important to acknowledge that upon completion of the placement of these platforms, a secondary process considers the placement of both collectable coins as well as obstacles to these platforms. The current geometry generator once again places control of this outcome in the hands of designers, by allowing for a platform to be considered or ignored for coin or obstacle placement.

CREATING PLAYABLE LEVELS

The generative process discussed in the previous section proves a sufficient model for the purposes of level generation: given that it crafts complete, finite and verified segments of play. Through use of the budget and constraint systems previously discussed, we can introduce an element of progression throughout gameplay. The *Sure Footing* game treats each generated segment of play as a *sprint* and is representative of the next 'phase' of the

[2] We use the term 'physical' to represent the playable manifestation of a desired action sequence.



(a) Budget Cost: 1

(b) Budget Cost: 2

(c) Budget Cost: 4

(d) Budget Cost: 5

Figure 6: A collection of hand-crafted adaptations of a ‘two path’ design pattern adapted from [4]. The budget costs are hand-tuned based on perceived difficulty to navigate each segment and will only be selected should the system have the available budget.

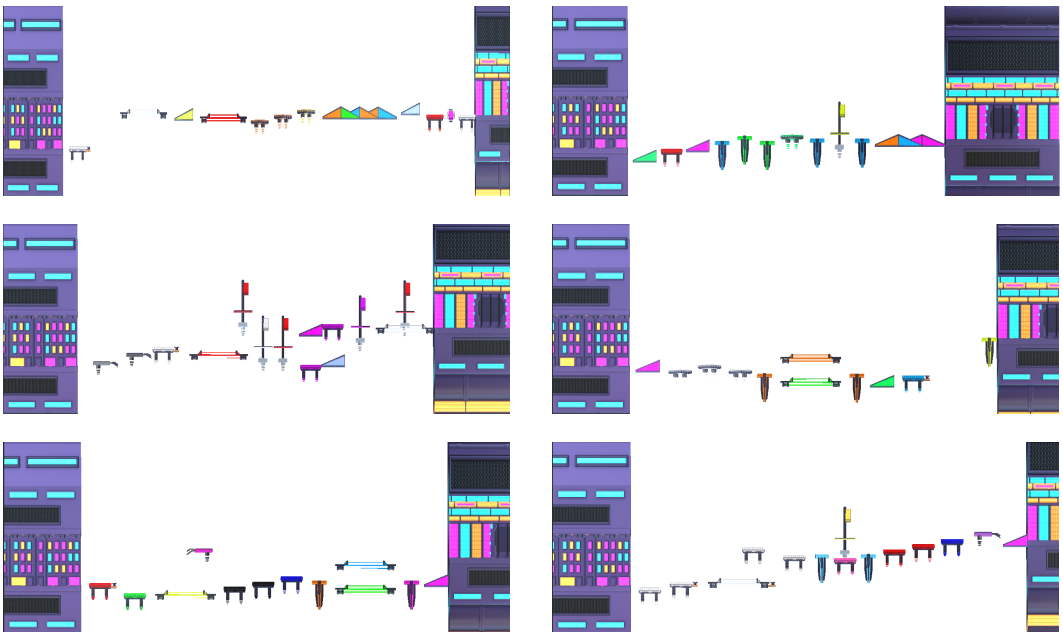


Figure 7: A collection of generated sequences placed into the *Sure Footing* game using the same budgetary constraints. Each generated *sprint* is bookended by a ‘rest piece’ designed to give players a small respite.

infinite running experience. This is segregated courtesy of in-game assets that are simple flat geometric planes: allowing the player a brief moment of respite from the challenges of the generated sprints. A collection of short generated sprints between rest-pieces is shown in Figure 7.

Table 2: Our collection of eight metrics adopted for expressivity analysis. These metrics are a combination of those adapted from existing research as well as some of our own design.

Level Generation Metrics	
Metric	**Description**
Leniency	A designer-defined model of the perceived 'difficulty' of a sprint.
Linearity	A linear-regression plotted against the generated sprint.
Length	The horizontal length of the sprint.
Verticality	The y-differential of the sprint with respect to its length.
Pattern Density	Number of actions generated with respect to action budget.
Geometric Density	Gameplay space between start and end points used by platforms.
Pattern Variation	The distribution of actions selected with respect to action budget.
Base Rhythm	The required jumps to navigate the actions generated.

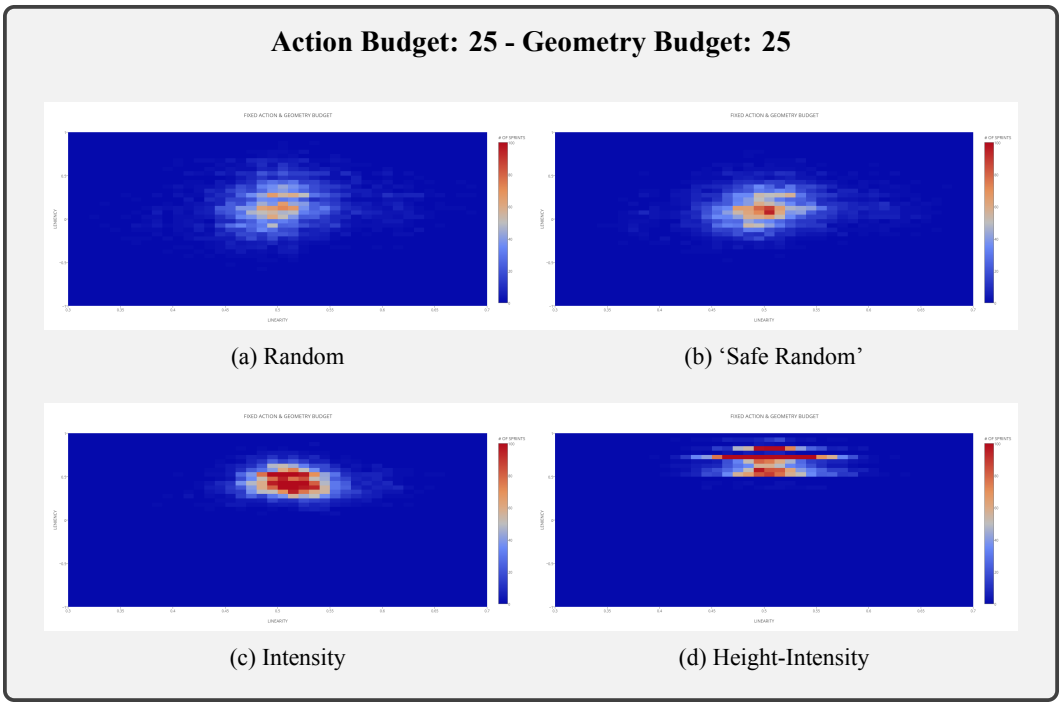
Given the fixed camera orientation obscuring future gameplay, new sprints are constructed at runtime upon reach the end of the current sprint. The flexible nature of the generators parameters allow for designers to tailor the constraints, action and geometry budgets and the currently active action and geometry generators at runtime. This permits a large amount of flexibility in not only the types of sequences a designer may choose to generate, but also retains an element of variability in the sequences crafted. There is evidence of this in the sprints shown in Figure 7, given that each permutation is the result of the same action and geometry budgets but yielding unique outcomes.

EVALUATION

Given we have established a framework for managing the scale whilst retaining variability, we need to appropriately evaluate the current state of the system such that we can state these assertions with confidence. As such, we look to the existing literature on quantitative analysis of generated platforming levels in an effort to identify the possibilities of the current system. We address this through the use of several metrics that reflect the variability and quality of the generated content.

We adopt the principle of 'expressive range' analysis first detailed in [18]: in which the authors comprise 2d histograms that identify the range of content that could be expressed. In order to achieve this we rely on a series of metrics found in Table 2, with the leniency and linearity metrics adopted from [18]. In addition, we looked at a number of additional metrics that can be adopted to quantify properties of the level generated detailed in [2]. As a result, all remaining metrics with exception of 'Geometric Density' are derived from that discussed in [2][3]. With our metrics defined, we aimed to quantify the expressivity of the system. For the purposes of this paper, we focus solely on the budgets afforded and generated 5,000 levels using the following configurations of the framework:

[3]However, it is worth noting that the 'Geometric Density' metric could be considered an inverse of the 'Negative Space' metric defined in [2].



(a) Random (b) 'Safe Random'

(c) Intensity (d) Height-Intensity

Figure 8: Expressivity models of linearity (x-axis) and leniency (y-axis) on the four action generators when operating under fixed action and geometry budgets.

Fixed Action Budget, Fixed Geometry Budget (Figure 8):
Recording the expressivity within a fixed action budget and geometry budget[4]

Increasing Action Budget, Fixed Geometry Budget (Figure 9):
Moving from a small to gradually increasing action budget but with a fixed (minimum) geometry budget.

Fixed Action Budget, Increasing Geometry Budget (Figure 10):
Maintaining a fixed action budget but gradually increasing the geometry budget permitted to the system.

The resulting measurement of linearity and leniency highlights a number of satisfying outcomes with respect to our level generation frameworks design. Firstly, the de-coupling of the action space with respect to the geometric construction of the sprint results in a large variations in the expressivity of the system. If we compare the use of the Random versus Safe-Random generators - Figures 8a, 9a and 10a versus Figures 8b, 9b, and 10b - there are distinct similarities, but the introduction of a handful of additional constraints forces the Safe-Random generator to be slighter more restrictive in the range of content that it creates. By contrast, the introduction of the Intensity and Height-Intensity based generators in Figures 8c, 9c, 10c and Figures 8c, 9c, 10c respectively, show a completely different expressive range. This is rather satisfying given that this is driven solely by changes made to how the

[4]The geometry budget was fixed at 50% between the the minimum requirement and the maximum possible given the generated action set.

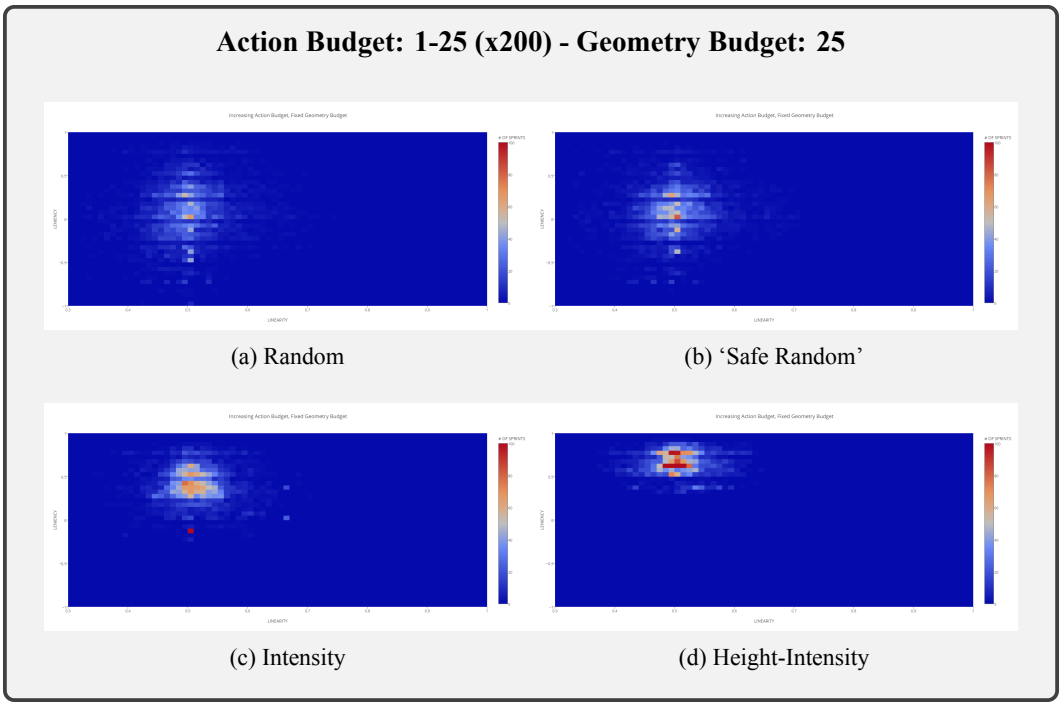


Figure 9: Expressivity models of linearity (x-axis) and leniency (y-axis) on the four action generators when operating under an increasing action budget, but with a fixed geometry budget.

action space is modelled. These results show that while these generators push the linearity to a more restricted range, it enforces what could be potentially more demanding levels given the leniency is scoped in the high-positive range.

While the results in Figure 8 provide sufficient evidence of the variability based upon action models, there is still need to consider whether the use of budgets influences the expressivity of the system. This is brought to our attention courtesy of Figures 9 and 10. In the former, the action generator budget is slowly increasing from 1 to 25. This has an influence on the amount of actions (and subsequently the generated space) the system will adopt. This is reflected in the expressive range of the system, which in many cases shows similar results in to Figure 8, but within a separate area of the expressive range. This would be expected given that the gradual adoption of more action budget would permit the system not only to create larger and more diverse levels, but also is influenced by the ability to use specific actions which may be beyond budgetary constraints on previous iterations. One notable issue that is apparent however is the relative similarities between the data presented in Figures 8 and 10. In the latter case, the action generation is operating on the same budget throughout, with the geometry generator receiving an increasingly larger budget. The resulting linearity and leniency is very similar, but with some fluctuations in the final outcome.

With this previous outcome in mind, we consider the results detailed in Figure 11 which highlights the expressive range of each configuration set with respect to the sprint length (x) and verticality (y) of the generated levels. It is clear from this set that the action budget

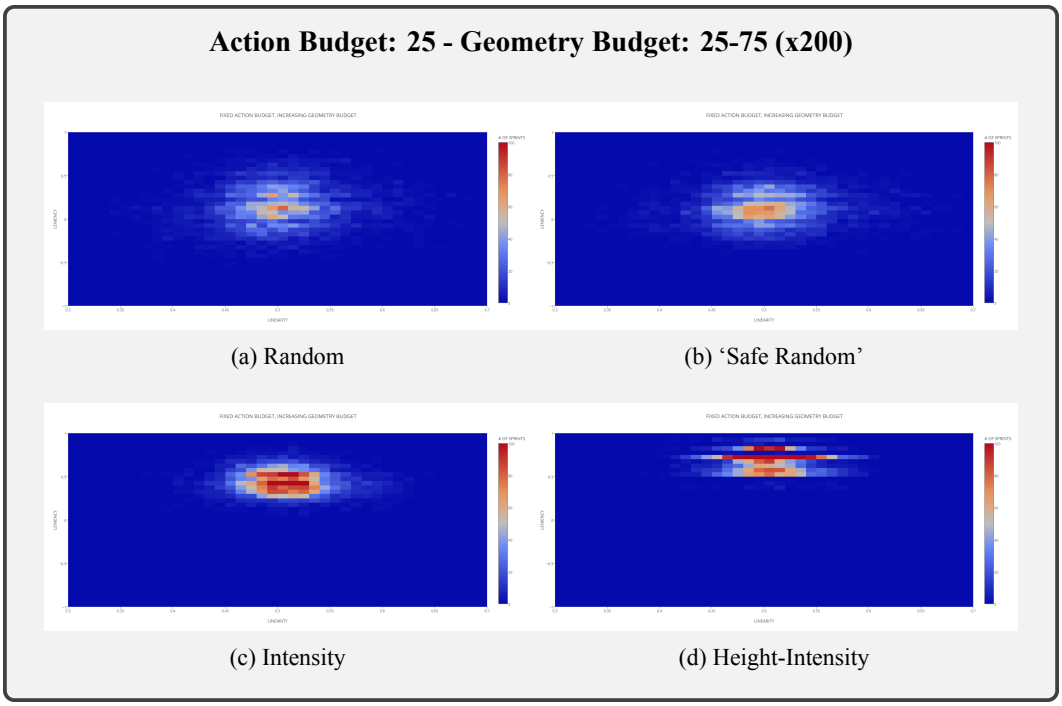


Figure 10: Expressivity models of linearity (x-axis) and leniency (y-axis) on the four action generators when operating under a fixed action budget, but with gradually increasing geometry budget.

has a large influence on the length of the levels created, with the verticality being influenced largely by the action generators unique behaviour. This is to be expected, given that as action budgets increase, the potential length of the level with increase as more actions can now be added. The increasing action budget ranges shown in Figures 11e through 11h reflect the range of these systems during gameplay, given the intention is to increase the budget over time as the player continues to progress. Meanwhile, the collection of results that highlight the increase of the geometry budget (Figures 11i to 11l) indicate that a larger quantity of content is generated within a fixed area. While at present we are only running on one sole geometry generator, it is still carrying an influence on the outcome of the generated content simply by modifying its permitted budget.

CONCLUSION

This paper presents a method for the creation of gameplay content that can be scaled courtesy of budget-driven components. The framework described in this paper is evaluated through its implementation as a level generator for an infinite runner video game. The two-phase nature of the generative system, combined with the use of budgets to curtail the generation process, provides for a wide variety of generated content that can be scaled by designers.

This is not to say that the work is without flaws or limitations: given that the generative processes are still markedly constrained despite their expressive nature. While some consideration need be given to the problem domain the framework has been tested within, there is still significant scope for further experimentation. The generative systems adopted largely

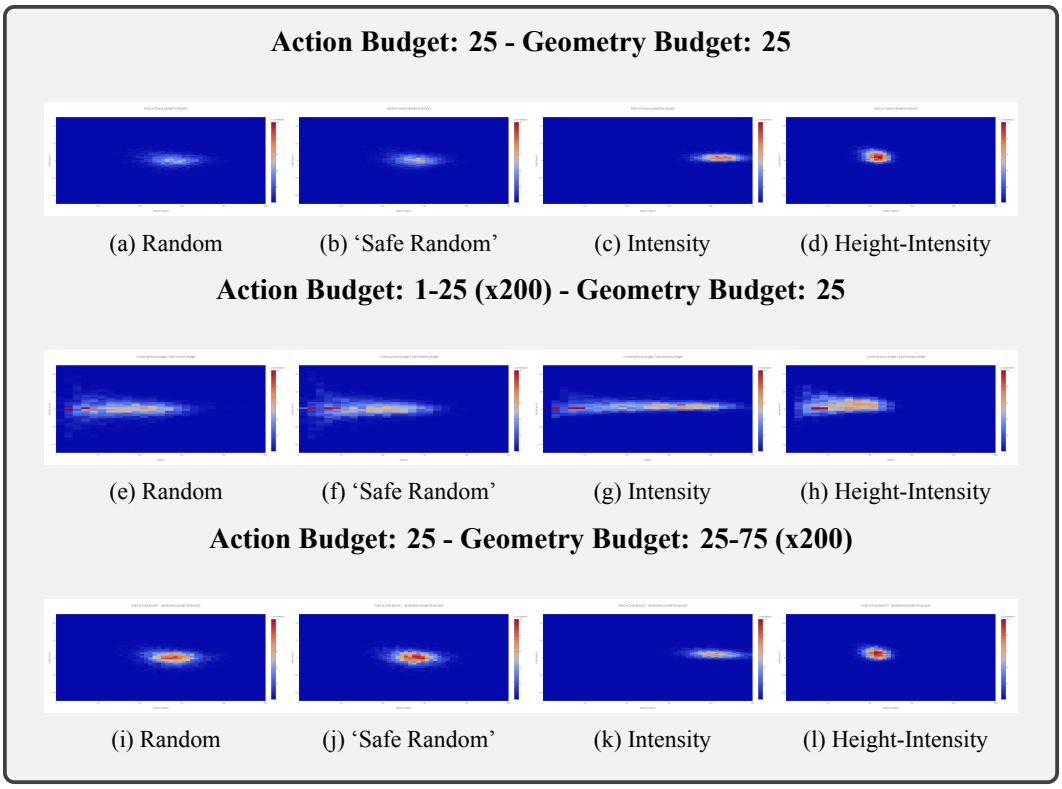


Figure 11: Expressivity models of the sprint length (x-axis) and verticality (y-axis) of the three level generation configurations.

adhere to the generate-and-test or constructive approaches of procedural generation as discussed in [22]. At this formative phase, the emphasis was on establishing the variety of generative approaches that could be mustered and how to maintain the scalability. As such, search-based generation was considered early on as an element of future work and is currently in development. One area ripe for future experimentation is the geometry generation component: given that its potential impact on the expressive range has not yet been explored. The results of our previous section would be influenced even more by the adoption of generators that manifest the actions in a variety of different forms. There is certainly work to be considered in level generation we referred to in our literature review, mostly notably those that adopt 'good design' principles to guide generation such as that found in [5, 20]. The authors are presently working on some expansions to the geometry generator that allows for greater freedom of expression. This is achieved through the combination of search-based procedural generation techniques, in conjunction with the introduction of new in-game assets that can yield more modular level constructions.

Furthermore, one as-yet unexplored aspect of this level generation framework is the opportunity for dynamic adjustment of parameters within the system. Considering the number of unique aspects of the generator that can be used to manage the scale and complexity of content, there is potential for a study in player modelling within the game. The notion of player modelling and analysis being used to influence content creation is well established within games research literature [3, 1, 17, 13], with notable works found in [15, 16] approaching

this in platforming games. While the level generator within the *Sure Footing* game aims to ensure a continued progression of difficulty and scale, the initial budget, the constraints employed and the rate at which budgets are increased and constraints relaxed would allow for a more adaptive and accommodating gameplay experience.

ACKNOWLEDGEMENTS

The authors wish to thank Rob Watling for his work on the level generation prototype and Molly Freeman for her significant contributions towards the art assets used in this project.

References

[1] Sander Bakkes and Joris Dormans. "Involving Player Experience in Dynamically Generated Missions and Game Spaces". In: *Eleventh International Conference on Intelligent Games and Simulation (Game-On'2010)*. 2010, pp. 72–79.

[2] Alessandro Canossa and Gillian Smith. "Towards a Procedural Evaluation Technique: Metrics for Level Design". In: *Proceedings of the 2015 Workshop on Procedural Content Generation*. 2015, p. 8.

[3] Darryl Charles et al. "Player-Centred Game Design: Player modelling and Adaptive Digital Games". In: *Proceedings of the Digital Games Research Conference*. 2005.

[4] Steve Dahlskog and Julian Togelius. "Patterns and procedural content generation: revisiting Mario in world 1 level 1". In: *Proceedings of the First Workshop on Design Patterns in Games*. ACM. 2012, p. 1.

[5] Steve Dahlskog and Julian Togelius. "Patterns as Objectives for Level Generation". In: *Proceedings of the 2013 Workshop on Procedural Content Generation*.

[6] Joris Dormans. "Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games". In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM. 2010, p. 1.

[7] Matthew Guzdial and Mark O Riedl. "Toward game level generation from gameplay videos". In: *Proceedings of the 2015 Workshop on Procedural Content Generation in Games*. 2015.

[8] Erin J Hastings, Ratan K Guha, and Kenneth O Stanley. "Evolving Content in the Galactic Arms Race Video Game". In: *Proceedings of the 2009 Symposium on Computational Intelligence and Games (CIG 2009)*. IEEE. 2009, pp. 241–248.

[9] Becky Lavender and Tommy Thompson. "A Generative Grammar Approach for Action-Adventure Map Generation in The Legend of Zelda". In: *Proceedings of the 2016 Conference on Artificial Intelligence, Simulation & Behaviour.* 2016.

[10] Becky Lavender and Tommy Thompson. "Adventures in Hyrule: Generating Missions & Maps For Action Adventure Games". In: *Proceedings of the 10th International Conference on Foundations of Digital Games. Playable Experiences Track.* 2015.

[11] Antonios Liapis et al. "Procedural Personas as Critics for Dungeon Generation". In: *Applications of Evolutionary Computation*. Springer, 2015, pp. 331–343.

[12] Peter Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common LISP*. Morgan Kaufmann, 1992.

[13] Noor Shaker, Georgios N Yannakakis, and Julian Togelius. “Towards Player-Driven Procedural Content Generation”. In: *Proceedings of the 9th conference on Computing Frontiers*. ACM. 2012, pp. 237–240.

[14] Noor Shaker et al. “Evolving Levels for Super Mario Bros using Grammatical Evolution”. In: *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE. 2012, pp. 304–311.

[15] Noor Shaker et al. “Evolving Personalized Content for Super Mario Bros Using Grammatical Evolution.” In: *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2012.

[16] Noor Shaker et al. “The 2010 Mario AI championship: Level generation track”. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 3.4 (2011), pp. 332–347.

[17] Adam M Smith et al. “An Inclusive View of Player Modeling”. In: *Proceedings of the 6th International Conference on Foundations of Digital Games*. ACM. 2011, pp. 301–303.

[18] Gillian Smith and Jim Whitehead. “Analyzing the Expressive Range of a Level Generator”. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM. 2010, p. 4.

[19] Gillian Smith et al. “Rhythm-based Level Generation for 2D Platformers”. In: *Proceedings of the 4th International Conference on Foundations of Digital Games*. ACM. 2009, pp. 175–182.

[20] Adam James Summerville, Shweta Philip, and Michael Mateas. “MCMCTS PCG 4 SMB: Monte Carlo Tree Search to Guide Platformer Level Generation”. In: *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*. 2015.

[21] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. “The 2009 Mario AI Competition”. In: *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE. 2010, pp. 1–8.

[22] Julian Togelius et al. “Search-based Procedural Content Generation: A Taxonomy and Survey”. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 3.3 (2011), pp. 172–186.

[23] Julian Togelius et al. “The Mario AI Championship 2009-2012”. In: *AI Magazine* 34.3 (2013), pp. 89–92.

[24] Michele Vinciguerra and Tommy Thompson. “A Procedural Generation Framework for a Robot Construction Game”. In: *Proceedings of the 7th IEEE Computer Science & Electronic Engineering Conference. Special Session on Computational Intelligence and Games.* IEEE. 2015.