Modelling Progression in Video Games

Pedro Pereira
Instituto Superior Técnico
Campus Taguspark
Avenida Professor Doutor Cavaco Silva
2744-016 Porto Salvo
Lisboa, Portugal
pedro.h.pereira@tecnico.ulisboa.pt

ABSTRACT
Games of today are extremely dependent on procedurally generated content to engage players in replaying a game. Most endless-running platformer games tend to generate their content based solely on the duration of the current playthrough, ignoring completely the skill of the player. We propose a model for progression in video games, using the level of mastery of the player when using the different mechanics and overcoming the different challenges provided by the game. The model is used to determine which features (mechanics, challenges) will be presented next to the player in order to maintain the player engaged and in flow. We theorise that this model allows for an increased level of replayability, fun and engagement in video games, thanks to the constant adaption of the game to the skill of the player.

This extended abstract follows the template used for the First Joint Conference of DiGRA and FDG.

Keywords
progression, player skill, level adaptation, procedural content generation, games

INTRODUCTION
Fun and engagement come with the progression of the challenges offered to the player and how these are overcome and interiorised by the player. Most single-player games lack replayability, which decreases the amount of fun and engagement a player can retrieve from a game, and fail to present an adequate level of difficulty for any given player. Usually they have different difficulty setting (easy, medium, hard) and always offer the same challenges to every player playing in the same setting. This often leads to mismatches between real player ability and overall game difficulty and breaks the feeling of progression, as players fail to find appropriate challenges to their skill set.

In this work, we will model the progression of a player in a game, not as a difficulty function, but as a ratio between the two dimensions of game flow, effectively bringing the skill dimension back into the problem of modelling the progression of a player. We will implement this model for a side-scrolling endless-running platform game. We theorise that this model allows for an increased level of replayability in single-player platform games and increased fun and engagement, because every consequent challenge is provided according to the current player skill within the game. Independently of their skill levels, our model seeks to make the players always feel challenged and engaged with the game. For this to happen, a custom editor tool was developed to allow for level designers to specify different adaption rules for the game, in order to guide the progression as the mastery of the player evolves.

RELATED WORK

We surveyed related work in *User Modelling* and *Procedural Content Generation* in games. In this section we will reference several previous works on how to specify game mechanics and how these can be used as building blocks of a game level. We will also present some works of level generation in platform games and how these can be related with emotions to provide automatically generated levels which expose the players to specific ones.

The Chemistry of Game Design
Daniel Cook

Cook (2007) proposes a way of modelling game mechanics using the game actions available to the player as entities that can be grouped into skill chains, showcasing the dependencies between these actions and the level of mastery of the player in each skill chain. This approach is useful as a tool to describe the full spectrum of game mechanics and how these are linked to create more complex actions. We intend to abstract the skill chain / tree technique and to relate it with the different features of a game: mechanics, challenges and paces. The evaluation for the mastery will also be revised in order to include the broader types of features we will use.

AI for Dynamic Difficulty Adjustment in Games
Robin Hunicke and Vernell Chapman

For our work we intend to control the demands of in-game obstacles according to the player's skill. We explore proactive adjustments to the procedurally generated playthrough, taking the overall performance of the player using specific mechanics to overcome specific challenges at a specific pace. We hope to develop a progression model with the ability of generating different playthroughs of the same difficulty, for players of the same skill levels, and be able to offer each player different playing experiences related to their own playing style, analogous to the *comfort* and *discomfort zones* discussed by Hunicke and Chapman (2004).

Modelling Player Experience in Super Mario Bros
Chris Pedersen, Julian Togelius and Georgios N. Yannakakis

The model (Pedersen et al. 2009) proposed by the authors was trained using different types of data:

- **controllable features**: parameters used for level generation, which affect the type and difficulty of the levels;
- **gameplay features**: features which depend on the player's skill and playing style, extracted from playing data logged during gameplay;
- **reported player experience**: a game survey was asked to each player after two game levels, each with different controllable features with the objective of making the players rank the games in order of emotional preference.

Since our tool allows a level designer to use the first two types of features to guide the progression of the playthrough and that the level designer can perform tests with users to evaluate each implemented solution, we can determine that all of three types of game features described in this model will be used to determine the player progression in our game, in order to procedural generate adequate playthroughs for the player.

Procedural Level Design for Platform Games
Kate Compton and Michael Mateas

This work (Compton et al. 2006) discusses *procedural level generation* in platform games as a sequence of automatically generated building blocks of different sizes. These methods allow for replayability of the game, due to the recycle of the different challenges that allow new compositions of challenges with a lower feeling of repetition. This approach will also allow us to grant a feeling of rhythm to the game, an important characteristic in games of the genre of endless runners. For the level generation, we will use a mix of the available paces, mechanics and challenges in the game, to provide compositions of challenges with a difficulty approximated to the current level of mastery of the player.

A Multi-level Level Generator
Steve Dahlskog and Julian Togelius

Dahlskog and Togelius (2014) discuss pattern-based level generation in platform games, using structures of different types and sizes as building blocks of a level. The presented method, which is an extension of a previous work by the same authors, solve the problem of progression and unity by using a new macroscopic structure which allows for a better-controlled difficulty curve. In our work, we intend to use different types of challenges, which we assume can be automatically generated, to construct a game level. We will assume that micro- and meso-patterns to be our challenges. Macro-patterns will be generated by concatenating a number of different challenges, with pre-determined distances between each one of them. Each one of these distances will be calculated from the levels of mastery associated with each one of the available paces provided by our game.

MODEL

Our model aims to evaluate the player's skill level with the mechanics, challenges and paces of a game in order to provide other mechanics, challenges and paces of the appropriate skill level of the player. We revised the original definition of *skill atoms* (Cook 2007) to include challenges and paces provided by the game, in addition to the mechanics (or skills) the player can execute. To better understand the differences between the different types of atoms, we need to define the following concepts:

– **Mechanic**: a skill that the player can execute in order to overcome a challenge; (examples: jumping, shooting, opening a door)

– **Challenge**: an obstacle that the game provides to the player;
(examples: ladder, powerful enemy, explosive barrel)

– **Pace**: the rate at which the game will provide challenges to the player; (examples: 1 challenge per second, 3 seconds after each challenge)

– **Mastery**: the level of proficiency and dexterity that the player has related to a specific mechanic, challenge or pace.
(examples: uninitiated, mastered)

We also revised and extended Cook's (2007) *levels of mastery* to include an additional *frustrated* level and revised the original definitions to better fit their usage with our different types of atoms:

– **Uninitiated**: atoms that never appeared to the player;

– **Initiated**: atoms that appeared to the player but still have a low level of mastery;

– **Partially Mastered**: atoms that the player is currently toying with, but has not yet mastered;

– **Mastered**: atoms that appeared several times and were successful overcome;

– **Burned Out**: atoms that the players have lost interest in exercising, because they appeared an elevated number of times and were almost always successful overcome;

– **Frustrated**: atoms that the players have lost interest in exercising, because they appeared an elevated number of times but were rarely successful overcome.

Every atom starts with the *Uninitiated* level of mastery, which changes to *Initiated* when it first appears. While the game is played, the internal value for the atom's mastery may change whenever the player is consecutively successful or not. The following figure represents the flow chart for the various levels of mastery.

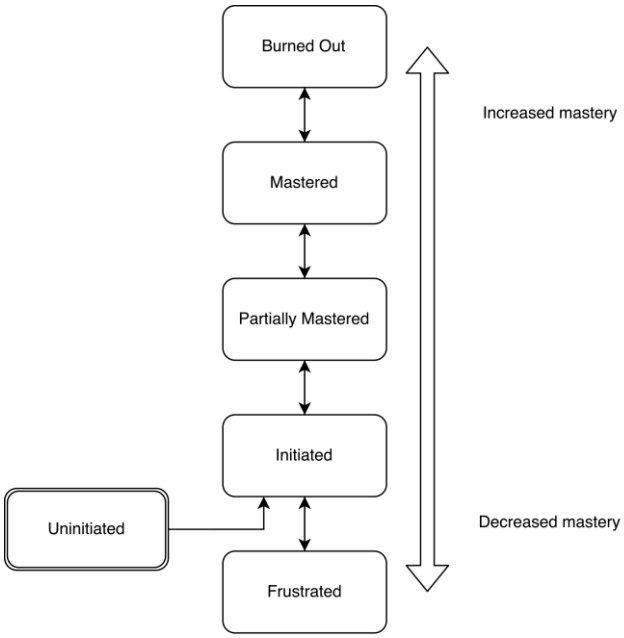

Figure 1: Flow chart of the different levels of mastery

Having defined how the different levels of mastery are related to an atom, we can now describe how these can be put together to guide the game to adapt its difficulty depending on the skill level of the player. In our graph, we will assume that an *active atom* means that what it represents may happen in the game. We can then use transitions to define when the next atoms will become active, based on the level of mastery of the current active atoms.

When the game starts, the graph will have only the initial atoms active. While the game is being played and the mastery of each atom changes the child atoms become active while its transition rule, based on the mastery of the parent atoms, is valid. Once these transition rules become invalid, the child atoms (and their respective children) will become inactive. The following Figure 2 represents a sub-graph unlocking several mechanics and challenges. Pace atoms are omitted for convenience.

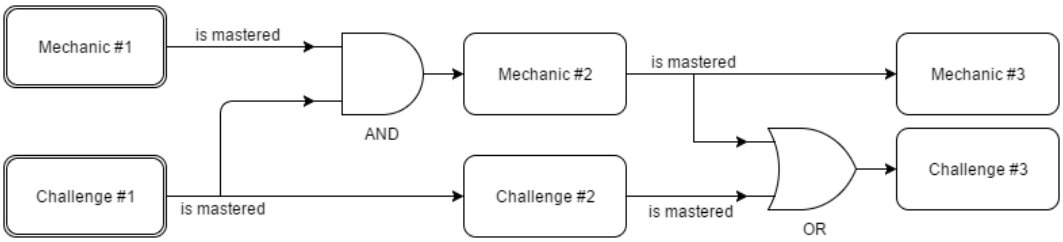

Figure 2: Example sub-graph enabling several mechanics and challenges in parallel

In this example, when Challenge #1 is mastered then Challenge #2 becomes active. To enable Mechanic #2 both Mechanic #1 and Challenge #1 must be mastered. To activate Challenge #3 only one of Mechanic #2 or Challenge #2 needs to be mastered. If for some reason the level of mastery for the Mechanic #1 decreases and stops being mastered, both Mechanic #2 and #3, and possibly Challenge #3, will become inactive.

Each mechanic, challenge and pace has its own window of attempts that registers the last *N* attempts related to that feature. Each one of these attempts has a different weight contribution in the totality of the window. The window of attempts is always ordered from oldest to most recent one. The weights for each entry in the window of attempts is used to specify which attempts will have a biggest impact in the calculation of the score. We assume that a success has a value of 1, while a failure has a value of 0. We can then calculate the average success of a feature using the following formula:

$$AverageSuccess = \frac{\sum_0^{nAttempts-1} attempts[a] * weights[a]}{\sum_0^{nAttempts-1} weights[a]}$$

Formula 1: Formula for the calculation of normalized scored based on a window of attempts

We can then map this score, in conjunction with the number of attempts in the window, to a usable level of mastery listed in Figure 1, as per the following formula:

$$Mastery = \begin{cases} BurnedOut, score = 1 \cap nAttempts \geq windowSize \\ Mastered, score \geq 0.9 \cap nAttempts > windowSize * 0.8 \\ PartiallyMastered, score \geq 0.5 \cap nAttempts > windowSize * 0.6 \\ Frustrated, score \leq 0.1 \cap nAttempts > windowSize * 0.8 \\ Initiated, nAttempts > 0 \\ Uninitiated, nAttempts = 0 \end{cases}$$

Formula 2: Formula for the calculation of a mastery based on a window of attempts

IMPLEMENTATION

Testbed Game

A custom graph editor was developed in Unity, based on the work of Levin "Seneral" G. (Seneral 2015), to facilitate the construction of different graphs. We can then connect this tool to a custom Unity game (also developed by us) and watch in real time how the mastery

for the different atoms evolve over time and what atoms are active at each moment, which tells us how the game is adapting itself to the progress of the player.

Figure 3: Example of a complete implementation of a progression model

Our editor supports 10 different types of game nodes (or atoms) and 5 helper nodes to implement more complicated logic for the transitions. The following table displays the organisation of the node selection menu for our 15 different nodes:

Boolean	Power	
	And	
	Or	
	Not	
	Memory	
Game	Mastery	
	Timer	
	Mechanic	Single
		Average Mastery
	Challenge	Single
		Spawner
	Pace	Single
		Spawner
	Pointer	Input
		Output

Table 1: Hierarchy of the node selection menu

Boolean Nodes

The nodes of this type are used to build more complex logic for the transitions between nodes.

Game Nodes

These nodes are used to enable/disable features of the game and to guide the model of progression, according to the graph implemented by a level designer. These are the most important:

– **single nodes:** used to allow and disallow the game to generate content based on a mechanic, challenge and pace;

– **spawner nodes:** used to specify the proportions between challenges and paces to decide which will appear in the game;

– **mastery nodes**: when connected to one of the *single* or *spawner* nodes, it measures the level of mastery (or the weighted average) of the specified masteries, challenges or paces. It is used to enable new paths of the progression graph when the player's skill is within a specified threshold in one or more features.

Testbed Game

In order to implement and test the model of progression defined in this work, we developed a custom game using Unity which directly connects to our node editor. This game is a side-scrolling endless-running platform game which features some simplistic mechanics (jump, double-jump, dash and slide) and a complementary set of simple challenges made specifically to test each one of the specified mechanics.

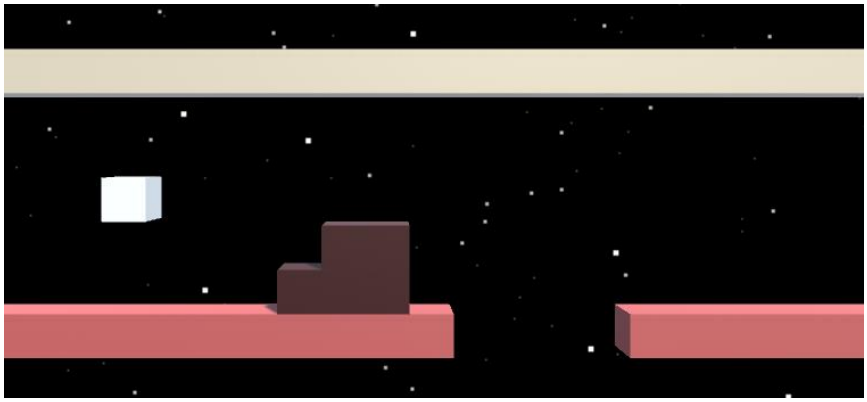

Figure 4: Current appearance of our game

The players plays as a white cube that must overcome all the challenges thrown at them with the objective of surviving for the longest period of time possible.

Player Mastery

As described in the previous chapter, we use Formula 1 to calculate the score of the player for a specific feature across a window of attempts, where we assume that each success has a value of 1 and each failure has a value of 0.

To decide if a given attempt is successful or not, the player must try to overcome each one of the features (challenge, mechanic or pace) in the game. The following figure shows the limits of each feature that are taken into account when deciding if an attempt on a feature was successful or not. For each type of feature, the condition to decide if an attempt was successful is different:

– **challenge:** an attempt to overcome a challenge is considered successful if the player doesn't collide with the challenge or falls through it (i.e. if the player can exit the vertical red limit , otherwise it is considered a failure);

– **mechanic:** an attempt to use a mechanic is successful if the player uses that specific mechanic when successfully overcoming a challenge (i.e. if a mechanic is used between the green and red vertical limits and the player exits the vertical red limit, otherwise it is considered a failure);
– **pace:** an attempt to overcome a specific pace is considered successful when the player successfully overcomes a challenge provided at that pace (i.e. if the player can successfully overcome all the consequent challenges with the same distance in between).

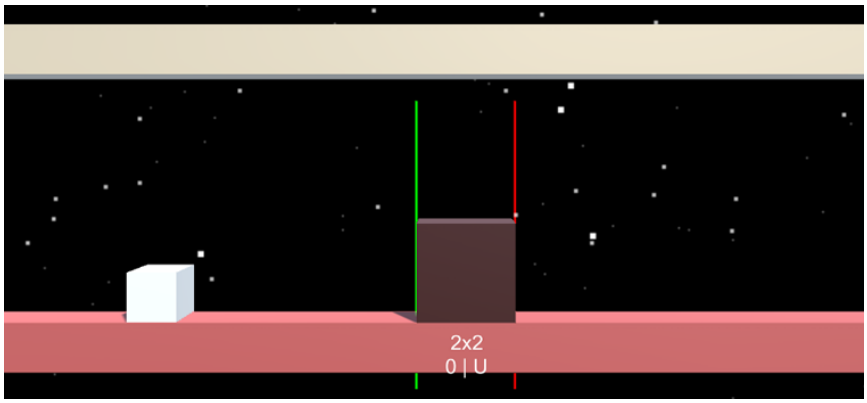

Figure 5: Start and end limits for an attempt on a 2x2 challenge

EVALUATION

In order to validate and improve our model and its implementation we evaluated the usability and quality of the implemented solutions by the testers, on various occasions during the development of this dissertation.

Usability evaluation

The objective of this evaluation was to visualise the interaction of each user with our tool and to determine usability issues that prevented or made more difficult the execution of each task. Our pool of participants were four non-professionals with zero or very limited experience in game development. We asked each participant to implement six tasks in our editor and we then analysed their interactions with the tool in order to gather feedback.

The participants noted some quirks regarding the node selection menu and the way transitions between nodes were drawn, which we fixed for the next evaluation session. Some participants also displayed some confusion after being told the description of the task. We revised in accordance the vocabulary used by us when teaching and showing participants the basics of the tool and when presenting them each task.

Qualitative evaluation

The objective of this evaluation was to measure the quality, efficiency and usefulness of the model, tool and implemented solutions for each task. Our pool of participants were three students of Game Design and three professional Level/Game Designers. We wanted to analyse the way professionals and non-professionals interacted with the tool in order to implement the same tasks. For this evaluation, we added a purely creative task so they could help us identifying limitations of the model or tool.

After all participants concluded their tasks, we were able to understand that professionals and non-professionals used the tool differently, although always implementing solutions with the same functionality. The non-professionals enabled and disabled features as specified in each task, while the professionals preferred to enable all *Challenge* and *Pace* nodes at the start and used respective *Spawner* nodes to control which features would appear in the game. The solutions also differed in which node output was used to disable another node. Non-professionals tend to block a node right after activating another one, as if it was a sequential operation, while professionals tend to block and enable nodes at the same time, as if it was a parallel operation.

After watching how the participants interacted during the new creative task and after performing a structured interview with them at the end of all tasks we learned that all participants were expecting a *multiple node selection* tool and a *clone node* feature. During the last task we noticed that the participants tend to copy parts of a solution to reuse it elsewhere in the graph. They expected these functionalities to be available in our editor tool, since they are available in most visual programming tools. The lack of these features harmed the efficiency and productivity of the participants.

Despite the aforementioned issues, all the participants provided positive comments regarding the tool. The most appreciated feature was the excellent mental picture of what happens in the game and the ability to visually debug in real-time each solution, while the game is being played. According to the users, this allows for easily check if the implemented model is valid and correct, and if any tweaks should be made to improve the experience for the player. All participants also referred that the tool is easy to master and has a very intuitive interface. It was easy for the participants to understand the differences and peculiarities between nodes and how they should be used. Although sometimes there were some questions regarding the functionality of a specific node, all participants quickly understood its function once they added it and tested it to see how it affected the game.

The professional participants gave additional feedback, due to their additional experience in game development and using visual tools of this type. They considered that this tool is very adequate for level designers, because they are used to use editor tools and visual programming. They also mentioned the modularity of the editor, meaning that the same final result could be achieved by different combinations of nodes and connections used, which is a good representation of the expressiveness of the tool. Finally, they appreciated the fact that our tool allowed the possibility of measuring burn-out and frustration levels of the player, which according to them is something difficult to measure in all types of games, and the fact that these “special” levels of mastery could also be used to implement a progression model for the player.

CONCLUSIONS AND FUTURE WORK

We started this project with the simple idea of bringing player skill to the dimension of Procedurally Content Generation (PCG), in a way that we could create a game without difficulty settings and that it adapted itself to what the player can do at each moment of a playthrough.

Both test sessions made us believe in the potential of our approach and our node editor tool. All the users were able to quickly understand all our concepts, to implement all the tasks they were asked to and were able to quickly find progression issues (such as when a graph path would never become active) and to suggest fixes and improvements to a solution in order to turn the playthrough of the game more interesting and enjoyable.

Also fortunate was the fact that all the users provided great feedback for features and improvements to be implemented in our tool that they felt would increase the productivity and expressiveness of each interaction, resulting in better solutions for guiding the progression of the player in each playthrough. We believe that this work was a good first step in the right direction to solve the problem of using player mastery in PCG and that we created a tool that can be easily adapted and used by level designers for generating content for other endless-running games.

We would like this work to incorporate changes that would extend the functionality of the editor in order to allow its use in other types of games. Our suggestions are: to extend the existent code base in order to facilitate the measurement of any kind of game metrics and respective nodes to use with our editor, such as time played, times lost, etc.; export of code stubs from existing canvas to allow the use of canvas created by our tool with other game tools and programming languages; additional tests with users using a quantitative approach, in order to validate and improve the current and to be implemented features of our editor and game; generalization of the work presented in this document, with the objective of testing the feasibility of using our tool with other types of games that could dynamically adapt its content based on the performance of the player, such as puzzle games - like *Angry Birds* (Rovio 2009) - or *run-and-gun* games - like *Metal Slug* (SNK 1996).

REFERENCES

Compton, K. and Mateas, M. "Procedural Level Design for Platform Games", AIIDE (2006), pp. 109-111

Cook, D. (2007) "The Chemistry of Game Design", in Gamasutra (19 July 2007) http://gamasutra.com/view/feature/1524 (accessed May 2016)

Csikszentmihalyi, M. "Flow: The Psychology of Optimal Experience" on Harper & Row, 1990.

Dahlskog, S. and Togelius, J. "A multi-level level generator", in Symposium on Computational Intelligence and Games (2014), IEEE, pp. 1-8

Hunicke, R. and Chapman, V. "AI for dynamic difficulty adjustment in games", in Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence (2004)

Nintendo (1985) Super Mario Bros.

Pedersen, C. and Togelius, J. and Yannakakis, G.N. "Modelling player experience in Super Mario Bros." in Symposium on Computational Intelligence and Games (2009), IEEE, pp. 132-139

Persson, M. (2008) Infinite Mario Bros. (Super Mario Bros. open-source clone)

Rovio Entertainment (2009) Angry Birds - videogame series

Seneral, Levin G. (2015) Node Editor Framework for Unity http://github.com/Baste-RainGames/Node_Editor (accessed May 2016)

SNK Playmore (1996) Metal Slug – videogame series

Togelius, J. and Kastbjerg, E. and Schedl, D. and Yannakakis, G.N. "What is Procedural Content Generation? - Mario on the borderline", in Proceedings of the 2nd International Workshop on Procedural Content Generation in Games (2011), ACM, p. 3