# Enhancing level difficulty and additional content in platform videogames through graph analysis

Fausto Mourato[1], Fernando Birra[2] and Manuel Próspero dos Santos[2]

[1]Escola Superior de Tecnologia - Instituto Politécnico de Setúbal, Portugal
fausto.mourato@estsetubal.ips.pt
[2]Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa, Portugal
{fpb, ps}@di.fct.unl.pt

**Abstract.** In this article we present a system that enhances content in platform game levels. This is achieved by adding particular gaming entities and adjusting their arrangement, causing consequent changes in the inherent difficulty and in path related aspects. This idea follows our prior work for the automatic creation of level environments. Starting with a primal level structure and a corresponding graph that sketches the user path, the system detects mandatory and optional path sections and adapts them in order to create more elaborate challenges to the user, forcing detours to gather specific objects or trigger certain events. Alternatively, a designer can create that base level structure and use the algorithm to adapt it to a certain profile. Also, some adjustments can be made to enhance multiplayer cooperative gaming for uneven skilled players, where the path is adapted to force a difficult route to one player and an easier one for the other player. Our experiments showed interesting results on some popular games, where it is possible to observe the previous principles put into practise. The approach is generic and can be expanded to other similar games.

## 1 Introduction

This article presents an approach to improve Procedural Content Generation (PCG) techniques used in the context of automatic level generation for two-dimensional platform videogames. Typically, in a platform videogame, the user controls the movement of a character (or various users control their respective characters, for multiplayer games) in a scenario, performing jumps to avoid terrain gaps and traps and overcoming opponents in a simple manner, such as jumping over the enemy or shooting him with a certain weapon. In the context of PCG, this type of videogames has been interesting to study because its mechanics raises several non-trivial questions on the process of automatic level creation. At first, it is important to ensure that level elements are positioned to provide a valid level where the user can fulfill a goal. In addition, the element positioning must make sense as a whole, resulting in a visually plausible scenario. Also, the level has to represent an appropriate rhythmic set of actions that the user should complete in order to keep him/her engaged. Finally, it is also important to take into account that those actions represent a challenge with a certain difficulty that has to be perceived and estimated.

adfa, p. 1, 2011.

In the scope of this work, the following platform videogames will be referred in order to support our tests and examples:

- *Infinite Mario Bros.*, an open-source *platformer* inspired by the classic videogame *Super Mario Bros.*, which has been used frequently in academia, namely in the development of intelligent agents to control the main character [14] and automatic level generation [9].
- The original version of the videogame *Prince of Persia*, which has unofficial level editors and technical details available online, as well as the recently released source code for the original *Apple* version of the game. This videogame has already been considered in research on the topic of computational complexity [15].
- *XRick*, an open-source remake of the original videogame *Rick Dangerous* which combines the free movement of *Infinite Mario Bros.* with closed environments similar to those in *Prince of Persia*.

The main motivation of this work is to provide content richness and personalization to platform levels that are created automatically in addition to simple linear gameplay. As we will see later in Section 2, where related work is unveiled, there are some interesting approaches on the subject of automatic level generation for platform games. Nevertheless, the main focus of the existing techniques typically goes to physical validity and definition of interesting sequences of challenges and actions without any particular meaning or context. This approach intends to be complementary as an additional step to any valid geometry generator, providing improvements in the content, such as:

- **Difficulty adjustment**, tuning the level to a certain player profile.
- **Challenge content improvement**, through the establishment of a non-linear path that consists of identifiable tasks such as, for instance, exiting the main path to grab a certain item and getting back to use it.
- **Inclusion of optional content**, as the system detects optional areas, which can be filled with bonus content defined by the game designer.
- **Multiplayer difficulty adjustments**, to adapt levels to be used simultaneously by players of distinct skills.

Our system starts with an imperfect level structure, which might be a basic geometry without additional gaming entities, and a corresponding graph. This graph is extracted from a list of rules that estimate the character's most relevant movements creating an approximate representation of the players' possible paths. Furthermore, the referred graph is analysed and every vertex is contextualized regarding its role on the objective of going from the starting point to the final position. After this graph analysis, a set of possible modifications is extracted. Some changes are iteratively selected from the referred set in order to reach a certain final value regarding level difficulty or length. Section 3 contains the details of this approach and explanations about how the system works. In Section 4 we will show some of the obtained results. An example case is provided and the generated changes are explained. Also, some performance

questions are analysed and the usage in other games is considered. Finally, in Section 5 we will present the main conclusions and guidelines for future work.

## 2 Related Work

Platform videogames, in particular in the context of automatic content generation, started to be studied in academic context by Compton and Mateas [1] with an analysis of the main components and the definition of a conceptual model to define this type of games. Later, a more detailed analysis has been presented by Smith *et al.* [10]. In this work, authors suggest a conceptual hierarchy to define the entities that compose a platform game level. Those principles were used in the definition of a technique to automatically generate levels based on the rhythm associated to player's input and actions [11], which has been applied in the prototype *Launchpad* [13]. The main goal behind this idea is to keep the user in a mind state that Csikszentmihalyi's referred as Flow [2], representing the ideal feeling of control and immersion over a challenging task. It can be seen briefly as a state in-between boredom and frustration, meaning that the task is, respectively, too easy or too difficult. Smith *et al.* also studied the expressivity of this approach [12] regarding linearity and leniency, an approximation to the concept of difficulty. Finally, following the same line of work, the system *Polymorph* presented by Jennings-Teats *et al.* [3] is an effort to adapt difficulty directly in the generation process as the player is moving forward in the scenario. The presented ideas provide an interesting way to generate good challenges but they tend to be mainly directed to the creation of straightforward games in open scenarios or simple casual game environments. Still, it is possible to have an initial generation step based on those principles and a further step to complement the content as the one proposed on this article.

Another important work to refer with similar features was presented by Pedersen *et al.* [8], focusing the already referred game *Infinite Mario Bros*. Levels are generated according to certain parameters. In particular, parameterization was applied to the following aspects: existing gaps in the level, average gap size, gap distribution entropy and number of direction switches. The main study focused on the concept of difficulty and possible adjustments to fit the user skills but, again, the generation process is directed to the construction of a sequence of jumps without a particular semantic meaning, preventing this method to be directly applied to other games.

A different approach was proposed by Mawhorter and Mateas [4]. The main principle is the composition of a whole level based on small pre-authored chunks, which can be assembled together. This allows a more varied set of outputs, depending on the number, type and variety of chunks that are considered. Although the authors also focused the game *Infinite Mario Bros.* we believe that this is a valid principle for other videogames, in particular for the generation of scenarios that represent closed environments with rooms interconnected with tunnels. These structures are likely to be obtained using merely small-sized chunks. For instance, the videogame *Spelunky*, a platform like adventure game, uses very similar principles and consists on that type of environments.

Another example of automatic level generation was proposed by Mourato *et al.* [6], using evolutionary computation, in particular genetic algorithms, to search for good solutions according to design heuristics applied to a fitness function and mutation and crossover operators.

One exceptional case that is directed to complementary content rather than simple movement and jumps was presented by Nygren *et al.* [7]. The authors proposed a method to integrate puzzle based content that requires the player to explore the level. It involves a three step process which consists, namely, in: graph generation, graph to level structure transformation and content creation. In the first step, a graph of possible positions is randomly created using genetic algorithms. That graph is then transformed in a valid geometry that fits those movements, with a search-based process. Finally and again with a search-based process, possible modifications are identified for each level segment from which one is selected to meet certain criteria. Once again, the experiments were directed to the videogame *Infinite Mario Bros.*. In some of the results it is possible to notice the existence of multiple alternatives and paths that lead to a dead-end, thus creating an exploratory theme. In our approach, as we will further see, similar situations are identified but, in this case, the adjustment algorithm takes advantage of those features to compose additional challenges and force exploration.

## 3 Iterative Content Adaptation Algorithm

In this section we describe the details of our technique. As stated, it works as a complement to a previously created level structure, which has to be valid regarding possible paths but that can be incomplete regarding gaming entities such as enemies, traps or doors. Therefore, we will start by presenting the initial conditions and requirements. Afterwards, we will cover the graph processing principles and the main steps on the level enhancement based on that processing.

### 3.1 Initial conditions and content

As previously referred, our algorithm works on a graph based analysis. This means that, in addition to the basic level structure, it is required to have a graph representation of that structure or a method to obtain it. In this topic, our approach works in the inverse order of the technique proposed by Nygren *et al.* that was previously presented, as we extract the graph from the geometry instead of creating the geometry to a certain graph. As previously stated, the main advantage of our approach is that we not only identify alternative paths and detours but also use them to place additional gaming entities to encourage exploration.

It is important to clarify that we are working with directed graphs, as some transitions might be unidirectional, such as the character falling into a hole, which has no way back. To provide a clearer notation, we will avoid mixing representations. Therefore, all edges will be considered to be directed and the cases of undirected edges connecting vertices $v_1$ and $v_2$ will be represented as a directed edge from $v_1$ to $v_2$ and another directed edge from $v_2$ to $v_1$. Besides the notation issues, in some cases this

distinction would be also mandatory because costs from $v_1$ to $v_2$ and from $v_2$ to $v_1$ might be different, as they may represent, for instance, a difficulty measure.

In our tests, we have been working with a grid based level representation where we have defined a set of pattern matching rules to extract the graph. In Figure 1, we present two examples of rules, composed by a pattern (on the left) and the corresponding graph entries (on the right) in the context of the game *Infinite Mario Bros.*. The complete set of rules is applied by going through the entire level grid searching for matches. Figure 2 shows a sample of the first level of the game *Prince of Persia* (on the left) and the corresponding graph (on the right) that will be considered on the analysis. That graph was obtained with the referred cell based pattern matching approach.

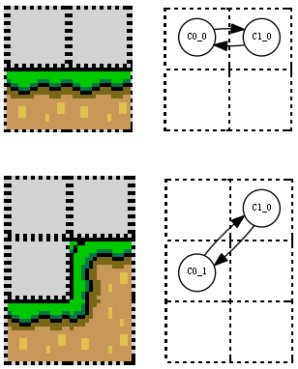


**Fig. 1.** – Example of two rules for graph construction, consisting of a pattern (on the left) and the corresponding graph entry (on the right).

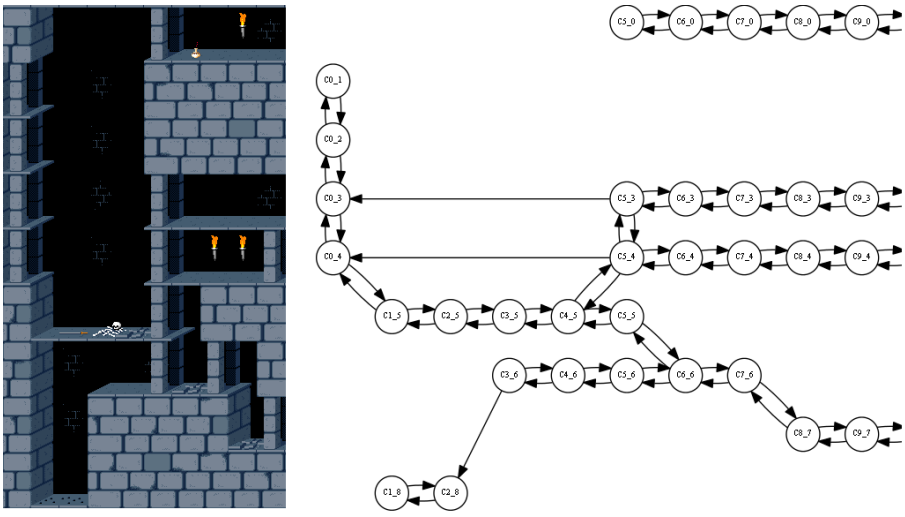


**Fig. 2. –** A sample of the first level of the game *Prince of Persia* (on the left) and the corresponding extracted graph (on the right).

The usage of this approach allows fast calculations over the level structure without requiring modeling all the physical rules implemented on the game. Still, alternative techniques that provide a similar graph structure might be considered. For instance, Mawhorter and Mateas' chunk based approach, presented on the related work, uses anchor points in the construction process that work as possible character positions. It is plausible to consider that those anchors can be used to extract the graph in a similar way as the solution that we have explained before.

Another aspect that is important to take into account is that graph vertices represent only spatial information and do not contain any additional information about the character or game state. We have also done some experiments with that in mind, considering that a level cell may provide or require a certain item. For instance, one cell may provide the user a key and another cell may represent a door that requires that same key. However, this tends to increase significantly the graph structure. For every character's reference position, all combinations of items in the character's inventory must be considered, which means that, considering an original graph with n vertices corresponding to the reference positions, we will have a new graph with $n.2^i$ vertices, with $i$ being the number of existing items in the level.

### 3.2 Graph initial processing

In order to reduce the computational effort of the algorithm, the first step is a graph compression that results on a reduced version with a smaller amount of vertices and edges. This compression involves removing vertices that are not significant to path computation processes and that can be seen as obvious intermediate steps in major transitions. In particular, the two following rules are applied:

- If a certain vertex $v$ has exactly one incoming edge $e_0$ (from $v_0$) and one outgoing edge $e_1$ (to $v_1$), this means that, regarding path calculations, $v$ is just a transitional step from $v_0$ to $v_1$. In this case, vertex $v$ and edges $e_0$ and $e_1$ are removed from the graph structure. A new edge is created directly from $v_0$ to $v_1$ with a cost corresponding to the sum of the previous costs defined for $e_0$ and $e_1$.
- If a certain vertex $v$ has exactly two outgoing edges $e_{O1}$ and $e_{O2}$ (to $v_1$ and $v_2$, respectively) and two incoming edges $e_{I1}$ and $e_{I2}$ from the same vertices, this means, in a similar way to the previous rule, that the vertex $v$ is an intermediate step in the connection between $v_1$ and $v_2$ in both directions. Again, this vertex is removed and the previous edges are also replaced by one edge from $v_1$ to $v_2$ with a cost value summing the costs associated with $e_{I1}$ and $e_{O1}$ and another edge from $v_2$ to $v_1$ with a cost that sums those from $e_{I2}$ and $e_{O2}$.

An example of the compression mechanism is provided in Figure 3, which contains a compressed version of the original graph represented in Figure 2.

Some restriction might apply in vertex removal. For instance, vertices corresponding to the start and end position should not be eliminated, as they have an active role on the level. This was implemented with the definition of a set of irremovable vertices.

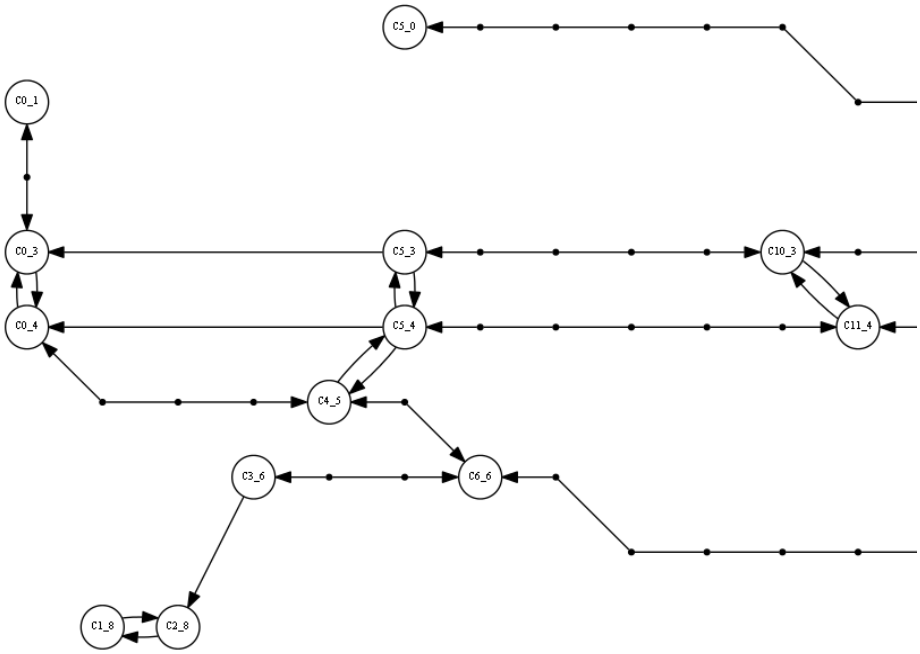


**Fig. 3.** – Example of a compressed graph. The compressed nodes are marked with dots.

In addition, some particular compression schemes might be deliberated depending on the game that is being considered. For instance, in the videogame *Prince of Persia*, it is common to have the situation represented on the left part of Figure 4, which will be represented by the corresponding graph that can be seen on the right part of that same figure. The triangular shape represented on the graph image by the three interconnected nodes could be merged into a single node without compromising path calculations. As another example, with a simple rule set, hill platforms in the game *Infinite Mario Bros.* will tend to create excessive vertical movement alternatives and prevent vertex compression, such as the case in presented in Figure 5. Vertices corresponding to the middle cells of the hill platform could be removed without compromising path analysis.

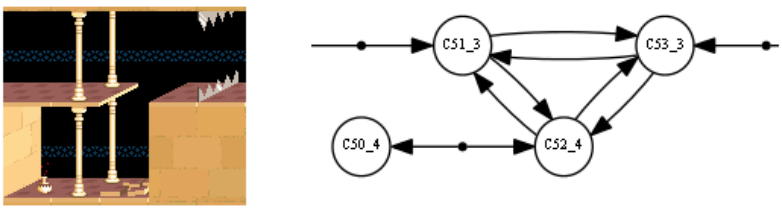


**Fig. 4.** – An example of a potential additional graph compression in the game *Prince of Persia*

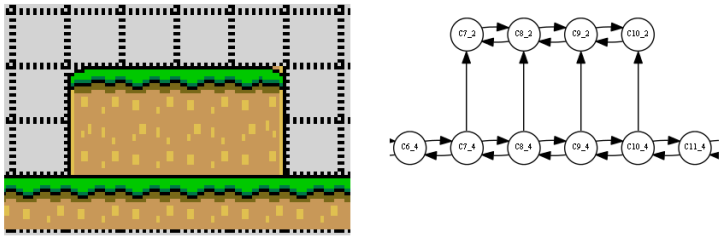

**Fig. 5. –** An example of a potential graph compression case in the game *Infinite Mario Bros*.

### 3.3 Path extraction and vertex classification

Having a compressed graph representing the level, the next step consists on categorizing the meaning of each vertex considering all routes between the start and the end of the level. The algorithm searches every possible path from the level entry to the level exit, using a breadth first search and ignoring all alternatives that visit the same vertex multiple times.

Following this step, every vertex is labelled with one of the following values:

1. **Mandatory**, if the vertex exists on all calculated paths;
2. **Optional**, if the vertex is only on some of the calculated paths;
3. **Dead-end**, if the vertex is not part of any possible path and has only one outgoing edge to a certain vertex and one incoming edge from that same vertex, meaning that if the character reaches that position he/she has obligatorily to go back;
4. **Unreachable**, if the position that corresponds to the vertex cannot be reached by the game character;
5. **Vain**, if there is no way back from that vertex to the main path;
6. **Path to Dead-end**, to all remaining vertices that, by exclusion, have the only purpose of providing passage from mandatory or optional vertices to a dead-end.

In addition, a tree is created representing path segments and alternatives for each segment, recursively. Leaf nodes represent trivial graph transitions. In Figure 6 we present, on the left part, a simple graph and on the right part, the correspondent tree that was created.

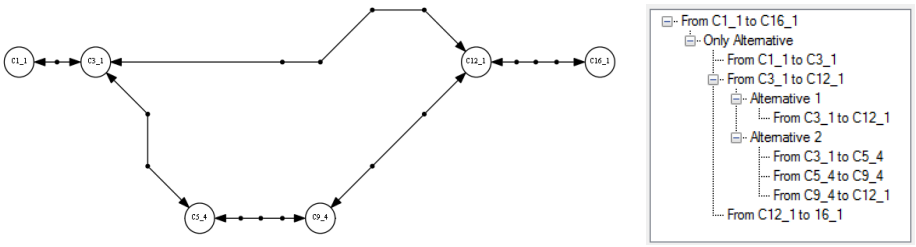


**Fig. 6.** – Example of a graph (on the left) and the corresponding segment decomposition represented as a tree (on the right)

Also, difficulty is estimated for each established segment, allowing a direct shortest path calculation even if difficulty in some segments is updated to different values.

### 3.4 Level completion algorithm

The level completion algorithm adjusts the level focusing on difficulty and similar related aspects. For the definition of the algorithm itself, the exact concept of difficulty may remain undetailed and be implemented as the game designer considers best. The key idea that has to be present is that difficulty arises from base geometry such as jumps over gaps, from gaming entities, such as enemies and traps, and from path structure, which forces the user to accomplish additional jumps and to encounter additional gaming entities. By design, the algorithm will have little control over jumps, as it would require graph recalculations, but will have high focus on the control of additional gaming entities and the definition of a composed challenge.

Difficulty has been characterized and estimated in distinct ways. It can be seen as a success probability [5] or a sum of coefficients [13], among other alternatives.

For the purpose of this algorithm, the requisites are the following:

- A higher difficulty level means that the level is more difficult than another with a lower value.
- Every level segment can be analysed individually to produce a difficulty value.
- A succession of analysed sections with certain difficulty values produces a final difficulty value for that succession.

In the tests that we will present, we have considered a sum of coefficients approach, where every graph transition contain a certain value that can be estimated by the game designer or mapped to users' performance after some gaming sessions. Moreover, when the path section presents two or more alternatives, the difficulty value that is considered is the lowest, as we assume that the user would pick the easiest possible solution.

The algorithm is able to apply the following changes to the base level:

- **Change the difficulty of a segment**, which is done by adding or removing an enemy or a trap or making a gap larger or smaller. Again, this is achieved with a pattern matching rule set. The graph is kept the same but a parallel data structure stores the entities that have been added to each section and the changes that have been performed.
- **Detour creation**, which consists on identifying a path to a dead-end from the main path, using the previously referred vertex classification, followed by adding a certain item in the path to the dead-end and making it required on the main path.
- **Cooperative two player game adjustment**, consisting on identifying two parallel alternatives for one particular section and using the previous two principles to adjust each alternative individually for each player. In addition, the game should contain a method to prevent both players to follow the easiest alternative.
- **Bonus entity addition**, which consists on adding collectibles or minor power-ups on certain dead-ends, creating secondary goals for the player.

The previous changes occur based on probabilistic coefficients that are established by the system after analysing the following values:

- Total desired difficulty value (whole level);
- Mean desired difficulty value (per segment);
- Player state estimator for game state on each graph vertex, detecting periods of possible boredom, flow or frustration.

The algorithm works iteratively in multiple adjustment passages, with the level difficulty being analysed for each passage. The process stops when it reaches the desired value or a limit on the number of iterations. At the end of each step, the previous features are analysed and the probabilistic coefficients are defined for each path section. For instance, if the level is too short then the detour creation probability in each section is increased.

## 4 Results

The next example shows how a simple level can be tweaked by the system. We will consider the game *Prince of Persia* in particular because of the possibility of adding gates and step switches to open those gates, which is an interesting implementation of the referred detour creation concept.

We have manually created the level structure represented in the background of Figure 7, which is just a simple draft. The level does not contain enemies, gates, switches or any traps. The starting point is the door on the left side and the level ends when the character reaches the door on the right. The system extracted and compressed the level graph presented as on overlay on that same image. Nodes have been named according to their coordinates on the grid, also marked in the image for convenience.

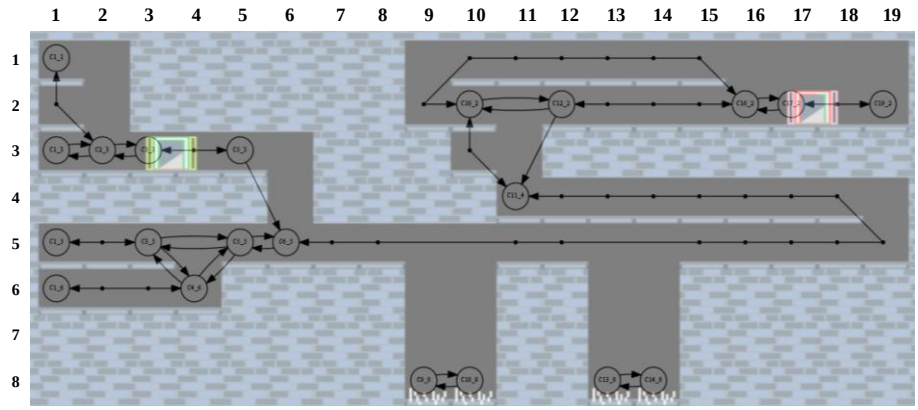


**Fig. 7.** - The Graph generated from the example level

The system classified the vertices as follows:

- **Mandatory**: C3_3, C5_3, C6_5, C11_4, C10_2, C16_2, C17_2
- **Optional**: C12_2
- **Dead-end**: C1_1, C1_3, C1_5, C1_6, C19_2
- **Unreachable/ Vain**: C9_8, C10_8, C13_8, C14_8
- **Path to Dead-end**: C2_3, C3_5, C4_6, C5_5

In addition, the tree presented in Figure 8 was calculated.

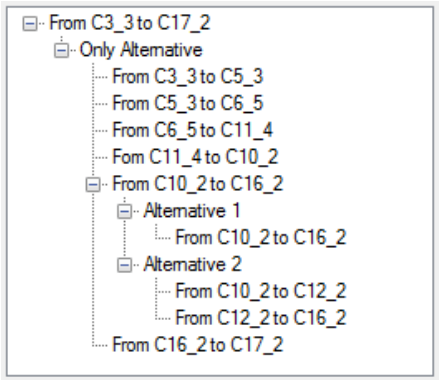


**Fig. 8.** – Calculated tree with all possible paths from the beginning to the end of the level

The system also calculated routes to the existing dead ends. In this case, the system identified the possible routes:

- C2_3 → C1_3
- C2_3 → C1_1
- C5_5 → C3_5 → C1_5
- C5_5 → C4_6 → C3_5 → C1_5
- C5_5 → C4_6 →C1_6
- C5_5 → C3_5 → C4_6 →C1_6
- C17_2 → C19_2

Considering our approach for estimating difficulty, a value of 55 was obtained. As a test, we defined the desired value of 100, so the algorithm was expected to increase the existing difficulty. One example run presented the result (printed to console) showed on Figure 9. The changes were then applied to the level, generating the content showed in Figure 10.

As we have stated, we have been testing this same approach in *Infinite Mario Bros.* and *XRick*. In the first, the detour principle is unlikely to be applied as the game does not have triggering events or object gathering. However, the primal tests related to difficulty and bonus content presented good results and the algorithm was able to adapt level segments. Coins are willing to appear as bonus on dead-ends and the number of enemies vary to match desired difficulty. On Figure 11 we show an exam-

ple of a randomly created level that has been perfected using our algorithm, matching a difficulty value defined by the user. As this game has a less restrictive set of movements, the entity placement is easier and less constrained. However, gap adjustments are hard to express because platforms can have different sizes and configurations.

```
Create Detour. Add button @ C1_1 and gate @ C5_3
Create Detour. Add button @ C1_5 and gate @ C8_5
Add guard @ C7_5
Add guard @ C17_5
Add spikes @ C18_5
Add guard @ C14_2
Add spikes @ C16_2
Final difficulty: 100
```

**Fig. 9.** – Example of a set of computed modifications

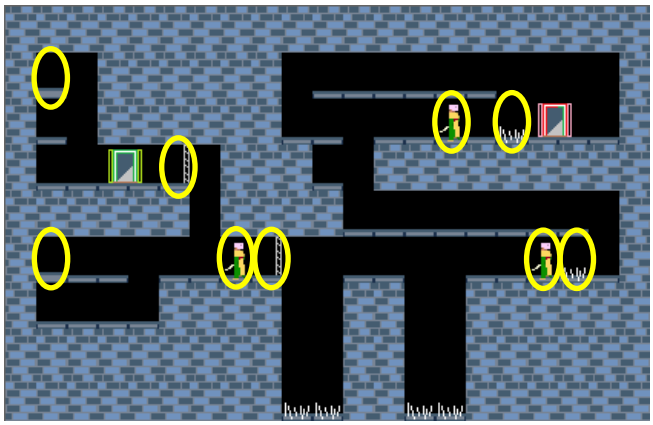

**Fig. 10.** – Example of a tuned level for the game *Prince of Persia* (changes are marked with ellipses)

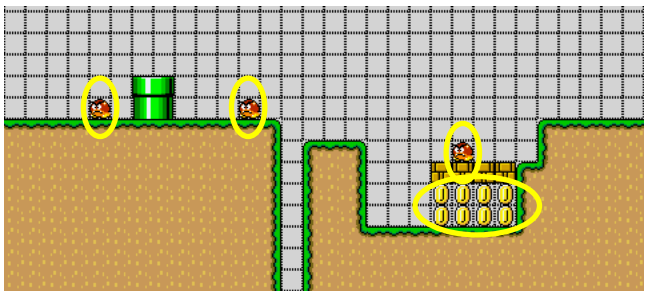

**Fig. 11.** – Example of a tuned level for the game *Infinite Mario Bros.* (changes are marked with ellipses)

Regarding the game *XRick*, primal experiments have also been done with similar results. This is a game with strong emphasis on triggers, where the detour principle is applied. In Figure 12 it is possible to observe a level that was created with that in mind. The main structure was manually created with two obvious dead-ends accessible with the ladders. The system automatically added the two guards on the bottom, the bonus sphinx on the left dead-end and a trigger on the right dead-end, marked with a stick, which removes the spikes on the bottom allowing the character to go through in the path from the left entry to the right exit.

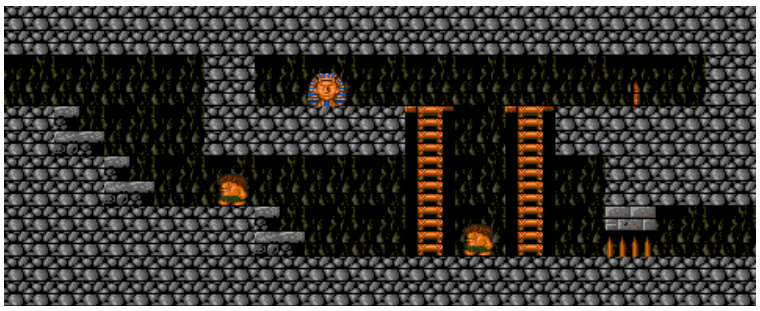

**Fig. 12.** – Example of a tuned level for the game *XRick*

## 5 Conclusions and future work

We have presented a technique that, considering a roughly defined platform level and a corresponding graph, finalizes the level adding optional content and adjusting difficulty. The presented approach is mostly suitable for games that have somehow in its mechanics the principle of gathering certain objects or triggering some events to unlock passages. For this reason we focused with more emphasis the game *Prince of Persia*, where we could see how our algorithm included new content. In addition, primal experiments with other games showed promising results.

Filling a level structure with additional content and adapting paths to force some particular actions produces improvements in content richness to the topic of level generation. This approach brings context to the actions that must be performed in order to capture the user interest. Also, the proposed multiplayer adjustment algorithm allows cooperative play in a shared environment independently of the player skills, promoting gameplay as an interpersonal experience. The considered games and most of classic platform games are only single player, by which this particular concept is still theoretical. Some experiments were done assuming possible versions of *Prince of Persia* and *Infinite Mario Bros.* supporting two players and the output appeared coherent. For instance, in *Prince of Persia* a set of gates and triggers are placed to force one player to follow one passage and the other to follow another passage. Consequently, one aspect to consider in the near future is to apply effectively these principles to a multiplayer *platformer*. Currently, we are directing our attention to the game *Open Sonic*, an open-source game based on the popular videogame saga *Sonic – the Hedgehog* which allows this type of principle to be applied.

The approach for extracting a movement graph based on rules using patterns is an alternative to physical simulations as it provides a faster way to analyse level content.

Another goal that has been defined to a near future is to export the generated levels into the games to have users to play them. At the moment we have achieved that goal with *Infinite Mario Bros.* and *Prince of Persia*.

## 6 Acknowledgments

This work was partially funded by *Instituto Politécnico de Setúbal* under FCT/MCTES grant SFRH/PROTEC/67497/2010 and CITI under FCT/MCTES grant PEst-OE/EEI/UI0527/2011.

## 7 References

1. K. Compton, M. Mateas. Procedural level design for platform games, *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE)*, 2006.
2. M. Csikszentmihaly. Flow: The Psychology of Optimal Experience. Harper Collins, NY, 1991.
3. M. Jennings-Teats, G. Smith, N. Wardrip-Fruin. Polymorph: Dynamic Difficulty Adjustment through Level Generation. *Proceedings of the Workshop on Procedural Content Generation in Games*, 2010
4. P. Mawhorter, M. Mateas. Procedural Level Generation Using Occupancy-Regulated Extension. *CIG2010 - IEEE Conference on Computational Intelligence and* Games, 2010.
5. F. Mourato, M. Próspero dos Santos. Measuring Difficulty in Platform Games. *Interacção 2010 – 4ª Conferência Nacional em Interacção Humano-Computador*, 2010.
6. F. Mourato, M. Próspero dos Santos, F. Birra. Automatic level generation for platform videogames using Genetic Algorithms. *ACE 2011, 8th international conference on advances in computer entertainment technology*, 2011.
7. N. Nygren, J. Denzinger, B. Stephenson, J. Aycock. User-preference-based automated level generation for platform games. *IEEE Symposium on Computational Intelligence and Games*, 2011.
8. C. Pedersen, J. Togelius, G. Yannakakis. Modeling player experience in Super Mario Bros. *Proceedings of the 5th international conference on Computational Intelligence and Games* (CIG'09), 2009
9. N. Shaker, J. Togelius, G. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Soreson, P. Pasquier, P. Mawhorter, G. Takahashi, G. Smith, R. Baumgarten. The 2010 Mario AI Championship: Level Generation Track, *Special Issue of IEEE Transactions on Procedural Content Generation*, 2010.
10. G. Smith, M. Cha, J. Whitehead. A Framework for Analysis of 2D Platformer Levels, *Proceedings of the 2008 ACM SIGGRAPH symposium on video games*, pp. 75-80, 2008.
11. G. Smith, M. Mateas, J. Whitehead, M. Treanor. Rhythm-based level generation for 2D platformers, *Proceedings of the 4th International Conference on Foundations of Digital Game*, 2009.
12. G. Smith, J. Whitehead. Analyzing the Expressive Range of a Level Generator. *Proceedings of the Workshop on PCG in Games, Monterey, CA*, 2010.

13. G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, M. Cha. Launchpad: A Rhythm-Based Level Generator for 2D Platformers. *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG)*, vol. 3, issue 1, 2011.
14. J. Togelius, S. Karakovskiy, R. Baumgarten. The 2009 Mario AI Competition. *IEEE CEC - Congress on Evolutionary Computation*, 2010.
15. G. Viglietta. Gaming is a hard job, but someone has to do it. *6th International Conference on Fun with Algorithms*, 2012.