Measuring Difficulty in Platform Videogames

Fausto José Mourato
Departamento de Sistemas e Informática
Escola Superior de Tecnologia
Instituto Politécnico de Setúbal
2910-761 Setúbal
fausto.mourato@estsetubal.ips.pt

Manuel Próspero dos Santos
CITI, Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica
ps@di.fct.unl.pt

Abstract

Automatic generation of game levels improves replayability and also allows content adaptation. One important aspect to take into account in the creation of any videogame is difficulty, in particular when it is possible to adapt content. However, defining difficulty is not a straight subject. In this paper we propose a metric for game difficulty in platform levels, mostly based on users' losing probability for each obstacle. This metric can be further used in automated processes that generate levels for this type of games, helping the process to recognize whether a level is suitable or not for a certain player. We also present some examples of the usage of this metric in commercial games.

Keywords

Human Factors, Platform videogames, difficulty measurement.

1. INTRODUCTION

Procedural generation of content is one active area in Computer Graphics with some common associations to other areas such as Artificial Intelligence and Human-Computer Interaction. In this paper we direct our main focus to the automated procedural generation of levels for platform games, aiming the contributions to understanding the related concept of difficulty. Considering a system that automatically generates platform levels, estimating and quantifying difficulty is useful because it improves the knowledge about the output.

The designation of platform games is used to describe a particular type of videogames where the user controls an avatar that has to accomplish a set of challenges. These typically consist of jumps between elements which are referred as platforms. Some popular examples of this type of games are: *Super Mario Bros*, *Sonic – The Hedgehog* and *Little Big Planet*. Screenshots of these three games are provided in Figure 1.

The main difference in generating automatically platform levels in comparison with other automated graphical generations, such as buildings, trees or even terrain, is that the final result represents a challenge with a certain difficulty, and a minor change can affect it with indefinite proportions. For instance, one platform moved slightly from its original place in a well balanced level can make it impossible to finish. Therefore, the main question that arises is: How do we measure difficulty in a platform level?

In order to answer the previous question, we propose a method to measure difficulty in a platform level, which we stated that can be considered as a relation to the probability of failure. Inside this topic our main contributions are:

- A study on the main questions related to difficulty in platform games.
- A proposal for a method to measure difficulty in platform game levels, relying on success probability.
- A practical usage of the difficulty proposed principle on some levels of existing commercial applications.

In the next section we will explain some of the major motivations behind difficulty measurements in the topic of automated generation of videogame content.

Figure 1 – Screenshots of the games *Super Mario Bros*. (A), *Little Big Planet* (B) and *Sonic – The hedgehog* (C)

2. WHY MEASURE DIFFICULTY?

As stated in the previous section, this study consists in improving the knowledge about difficulty in platform games, which can represent a step forward in processes that automatically generate this kind of levels. There are several possible approaches for platform level generation, but only a few have been made effective, as we will see in a later stage of this document, in section 5, where the related work is presented with more detail.

To better understand the problem, we can consider a system that generates levels on-the-fly where the user can play. For the sake of ease we can also consider by now that the system validates physical restrictions and thus produces always valid levels. This means that the user will always have a chance to succeed, depending only on his/her skills. However, even in these conditions, there is no way to ensure that the challenge is appropriate for the user because all levels are treated equally. This leads us to the need of matching users' skills to provided challenges, which should have an appropriate dimension of difficulty. As a result, it is required that generated levels have a certain difficulty value. In the next section we will look at the most common situations that are present in most platform games in order to understand where the difficulty lies and hence measuring it. In section 4 we will also present some examples to show that several common game situations can be fitted in our reference situations.

3. DIFFICULTY IN PLATFORM GAMES

3.1 Understanding difficulty in a whole level

We presented the problem of measuring difficulty on platform game levels and its importance for automated procedural generation, in particular for content adaptation. In this section we will try to answer the following question: What makes a level difficult?

In order to understand where difficulty lies, our main examples will consider the more trivial cases for platform levels and their possible adaptations, covering the common structures that this type of games have, as proposed by Compton and Mateas [Compton06]. Levels are defined with one start point and one end point in a bi-dimensional geometry. Some levels may have certain objects that harm the player's avatar causing him to lose. These will be represented as spikes, which are common objects in this type of games. In addition, there are two main considerations to take into account regarding to the user experience, which are the following:

- The users are aware of the task they are doing, which means that every challenge is directly identifiable, as proposed by typical usability heuristics such as those presented by Desurvire et al. [Desurvire04].
- Trivial tasks, such as moving from one place to another in a single platform or climbing up a ladder, are not relevant in difficulty measurements.

With those principles we will consider the five different cases represented in Figure 2, which can be briefly described as follows:

A. A straight way where it is impossible to fail. Since the user knows what to do he/she will not run blindly against the left wall.

B. An impossible level because the end point is physically unreachable.

C. An almost impossible level with the end point being almost physically inaccessible.

D. One level with a single jump that the user must accomplish to reach the end point. Failure attempting the jump will cause the player to lose.

E. A level where the user needs to jump from one block to another but if he/she fails he/she will be able to try again after passing through another obstacle that may cause him to lose. The stairs represented in this case simply mean that climbing to the first platform is a trivial action that every user can do without difficulties.

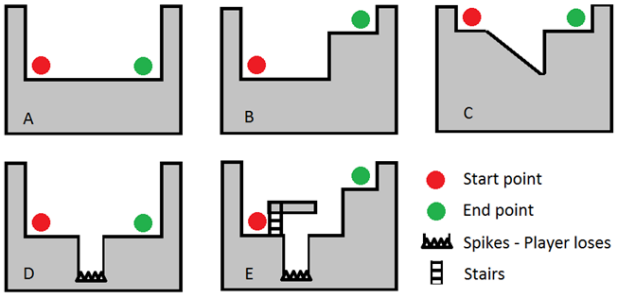

Figure 2 – Five examples of common situations in platform games

The first case is a generalization to represent the easiest level that can exist. Due to the subjective nature of the topic this cannot be proven but common sense seems to be enough for this situation. The question here can be reversed and be thought with the following question: Can we design an easier level than this? In fact, the task required to complete the referred challenge can be described as a single action, where there is no possible way of losing. Simplifying the problem, it is plausible to say that this first example is easy because the probability of losing is zero. Completing this level consists on doing one simple task with a probability value that can be considered one. A simple graphical representation is available in Figure 3.

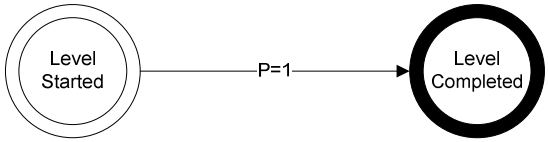

Figure 3 – State chart for example level 1

The second example tries to create a contradiction with the conclusion that we can retrieve from the first example. In this case, the probability of losing is again zero, because there are no elements to harm the player's avatar. However, winning probability is also zero because the user needs to jump to an unreachable place. The same problem will occur if we think of an adaptation of the first example in which the start and end positions are too

far apart that there is no user with enough patience to complete the challenge. This leads us to another aspect to take into account that is the probability of renouncing. Considering the psychological approaches to this subject, the user will resign playing if he/she is bored or frustrated. Boredom will occur in a situation as the adaptation of the first example, with an almost infinite distance between start and end points. Frustration will occur in situations as the second example, where the user in presented with an impossible task. The idea of a good game is to keep the user in-between those two states, in a state that is designated as flow [Csikszentmihaly91]. The state chart in Figure 4 is a possible representation of the described situation.

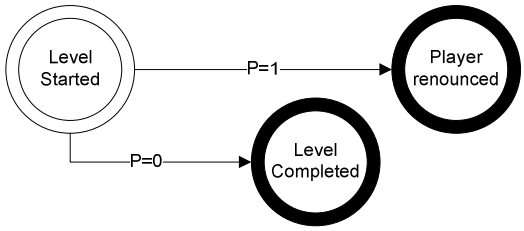

Figure 4 – State chart for example level 2

In the third example we keep the user again in a harmless level with one single jump that has to be accomplished. The main difference is that in this case the full probability is divided in success and giving up. A state chart is provided in Figure 5 to represent this situation, considering $P(j_1)$ the probability of success in the first obstacle, in this case a jump, and $P(r)$ a general resign probability for the user after failing one challenge. If we consider an impossible jump, it is possible to state that this case and the correspondent diagram will result in the previous situation. In the next subsection we will explore the associated calculations that can be extracted.

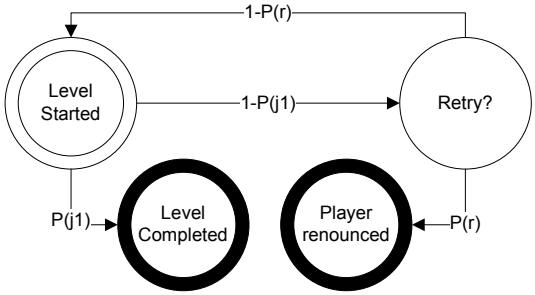

Figure 5 – State chart for example level 3

The fourth example adds the existence of harm, which cause the player to fail immediately. Considering that the distances between the start and the end point are small, we can consider that the user will not forfeit because the level consists in one single task. So, the probabilities will be divided directly in winning and losing, as represented in Figure 6. Those probabilities are only linked with the difficulty of the jump, which will have a probability of success defined by $P(j_1)$ as in the previous example. We will also approach this problem with some more detail in the next subsection.

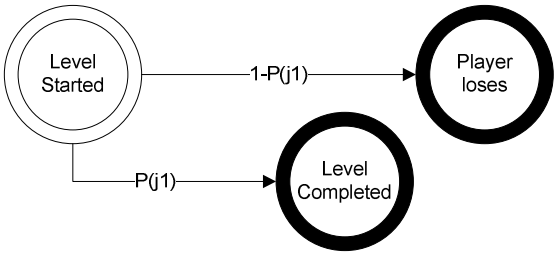

Figure 6 - State chart for example level 4

Finally, the fifth example gives us a single jump with no direct harm that causes failure but that requires succeeding in another jump in order to retry. The result is a loop as in the previous case. The main difference is that, in this last example, the retrying loop can be broken either by renounce or by failing in the process. This is represented with the state chart in Figure 7.

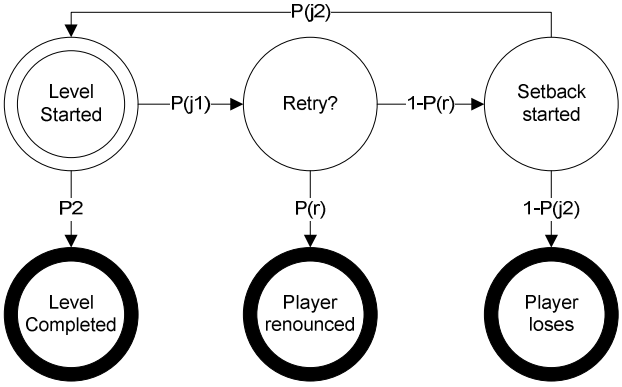

Figure 7 – State chart for example level 5

After the previous examples we can identify two main variables that contribute to difficulty in one level:

- $P(r)$: The probability of renouncing after failing one attempt, which we will consider constant in one level. This value is directly related to the player's resilience.
- $P(j_i)$: The probability of achieving success in the jump or similar challenge with the index *i*.

Also, we could identify that there are two main elements to consider for the difficulty calculations over one level, which are the obstacles where the avatar may directly lose and the obstacles that will make the avatar repeat other obstacles that may eventually cause renounce or failure.

With these concepts in mind, we will proceed to the next subsection to understand how to extract final probability values and thus difficulty.

3.2 Measuring difficulty in a level

In the previous subsection we represented the most common situations in platform games and identified that difficulty is related to the probability of being unsuccessfully, which might be caused by failure or resignation. The studied situations were represented as state charts in which every transition is associated with a probability value. Our proposal is to calculate the probability of suc-

cess and failure that is recursively represented in those charts, and thus extract difficulty.

Calculating probabilities in cases like one, two and four are simply a sequence of independent cases of probabilities, thus measuring difficulty can be calculated by multiplying all the values from the start to the end point. Cases like the examples three and five have loops in the representation, which makes the calculation less simple. For instance, probability of success in example four is the probability of making the jump on the first attempt plus the probability of making it at the second attempt, if there is a second attempt, and so on. This can be expressed mathematically with an infinite equation to represent the probability of success $P(s)$ as follows:

$$P(s) = P(j1) + P(j1').P(r').P(j1) + \cdots + \left(P(j').P(r')\right)^{n}.P(j1) + \ldots$$

This equation can be compressed, resulting in the following:

$$P(s) = P(j1)\sum_{i}\left(P(j1').P(r')\right)^{i}$$

If we think of a whole level, measuring these probabilities with a formal model will require a long set of equations and the calculation of these series. However, the information is more likely to be represented as a state-chart or a graph rather than in mathematical notation, so these calculations might be done with a recursive and iterative algorithm that spreads probabilities over the nodes until a certain precision is reached.

Considering the previous representations for the whole problem and that we can recursively spread probability values through a graph in order to understand what are the final probabilities to achieve success and failure in one level, it is important to extract the probability values for the connections in the graphs, which represent the difficulty for isolated jumps or, more generically, isolated obstacles. This topic is about to be approached in the next subsection.

3.3 Difficulty for independent obstacles

In this subsection we propose a method for measuring the difficulty for each isolated jump or challenge, which can be used to feed the probability graphs presented in the last subsection.

One simple assumption is that a single jump is more difficult to accomplish if the gap to jump is bigger. This is valid for platforms at the same height, but for different heights the problem is not that direct. With a physics simulator it is possible to identify the multiple alternatives to make a successful jump between two platforms. Compton and Mateas [Compton06] already proposed this approach but only as a theoretical possibility without concretization. In addition, looking at all existing combinations of jumps also gives just a notion on the difficulty of a jump by counting the successful and failure cases. Some of those cases will be more likely to happen than others, because the user keeps correcting the movement in the jump. Since transposing physical constraints to a difficulty measurer may be a difficult task, especially if we are estimating difficulty in an existing game with closed source, we propose a faster alternative based on the jump characteristics. So, in a jump, at a higher level of abstraction, the player is trying to jump from one point to another. These points represent the edges of the origin and destiny platform. With this in mind, our approach consists in launching a projectile from the first point P_0 and measuring a possible margin of error for its trajectory relatively to the second point P_1.

Recalling some basic physics, a projectile trajectory can be defined by the following set of equations in order of time: $x(t) = x_0 + v_{0x}.t$ and $y(t) = y_0 + v_{0y}.t - \frac{1}{2}a.t^2$

Once every jump is relative to the platform from which the user is jumping, we can consider the origin point (P_0) as the reference to simplify the equations. Also, we will not consider a throwing angle, so the initial speed can be defined as a constant (K_i), configurable for different games. Finally, these calculations are space oriented and the jump duration is not used, so the calculation can be resumed as one expression, which is basically a quadratic equation intersecting the origin, simply defined by:

$$f(x) = K_i.x - \frac{1}{2}a.x^2$$

This trajectory can be seen as the biggest possible jump the avatar can perform. By intersecting the projectile with two lines that are parallel to the Cartesian axis and that also intersect the destiny point, we can estimate the possible deviation to the trajectory that is still a valid jump. From now, this deviation will be referred as the *error margin*. In fact, we are identifying two values: the height of the player when he/she horizontally reaches the platform (Δy) and the horizontal amplitude of the jump (Δx). So, our error margin has values in x and y axis (m_x and m_y) defined by $\Delta_x - x_1$ and $\Delta_y - y_1$, respectively, with $P_i = (x_i, y_i)$. This concept is graphically summarized in Figure 8.

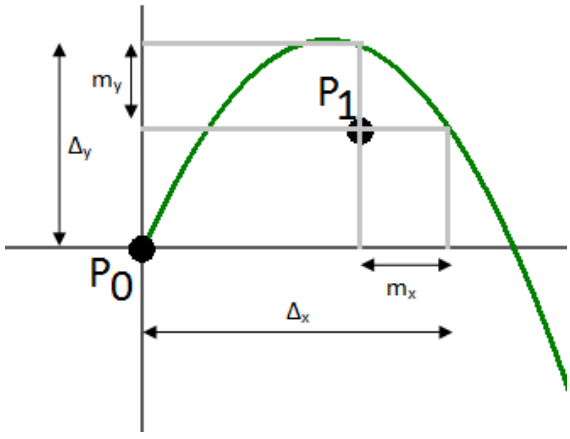

Figure 8 – Graphical representation of the concept of error margin

Obtaining m_y requires knowing Δy, which consists on calculating $f(x_1)$. A negative value means that the platform in unreachable with the reference jump. Here, the system has to identify whether the platform is really unreachable and thus the error margin is immediately zero without any other calculations, or the jump has to be attempted differently (for instance, moving the origin point

to the left). For positive values we measure the horizontal tolerance by calculating the value of x for which f(x) equals y_1, which consists in solving a second degree equation and selecting the appropriate root. To normalize results we consider the error margins as relative values to the full distances measured along the axis. In the end we multiply both the normalized error margins to define the final error margin for the obstacle (M), defined between zero and one.

Empirically, one can perceive that difficulty is related to the error margin in a non-linear fashion. As error margin decreases it is expectable to have a few failures occasionally occurring, but as the error margin gets lower, more and more failures are willing to occur. To reflect this principle, we represent the final difficulty value for each obstacle based in an exponential function with the following equation for difficulty in the obstacle of index *i*:

$$\mathrm{D(i)} = M_i^{Kd}$$

The constant Kd represents the mapping of the linear error margin and the expected exponential distribution that can be configured to a certain player profile. In the results presented in section 4 we will present some comparisons.

3.4 Dynamic entities

Our main principles consider a static scenario, which is somehow a limitation. Some alternatives and proposals for the dynamic question will be addressed in future work. However, in this subsection we will present how we deal with this problem in our current studies.

Typically, in a platform game, dynamic objects are moving platforms and enemy creatures. For the first ones, we can consider a gap that separates two platforms and that is impossible to jump across the gap. One moveable platform slides from one side of the gap to the other, so the user needs to use it as a transport. Considering that the user is focused in the task and it is not exploring the scenario in a way that is not intended, this process consists in jumping two gaps. The main issue is that the movement of the dynamic platform produces an extra error probability. Also, the faster the platform moves, the bigger the error might be. So, for situations like this, we apply the calculations previously presented for normal jumps. However, to represent the increase of difficulty caused by dynamics, the gap distance to consider is a normal jump to the platform at the nearest place plus an extra term that is proportional to platform speed and a constant. Representing enemies can also be considered as jumping implicit gaps. Depending on the enemy's characteristics (mainly speed and size), we manually define a value for P_1.

3.5 Final remarks

In this section we have presented an approach to measure difficulty in platform game levels. For each obstacle we defined an error margin to extract an estimated probability of succeeding. The whole level can be considered as a graph with multiple transitions where the referred probabilities can be applied to obtain a global value. The next section presents some results about applying this concept.

4. CASE STUDY

In this section we will briefly explain our experiences on measuring difficulty in levels of existing commercial applications. Our experiments consisted on the following:

- Mapping existing game levels in a representation with only the situations previously presented and measuring difficulty in the mapped levels with the principles previously presented.
- Testing levels with real players on the referred levels, measuring their probability of success and comparing to the estimated results.

4.1 Mapping and measuring existing levels

As we previously referred, we applied our proposed metrics to some existing games. The classics are useful because normally it is possible to find online bitmaps with whole level representations. The results of this subsection present difficulty measurement in some levels of the game *Super Mario Bros*.

First, in Table 1 we present the success probability for a regular user in some levels of this game measured as we proposed in this document. We selected a set of levels that share similar objects and obstacles to avoid having particular aspects biasing the conclusions, such as the existence of *bosses*, portals or different physical conditions (for instance, some levels are played under water). The value *T* in the table represents the number of times the user will retry an unsuccessful jump.

Level	T=1	T=2	T=3	T=∞
W1L1	10.8%	32.4%	40.8%	44.6%
W1L2	3.5%	10.7%	13.7%	15.0 %
W1L3	.5%	.8%	.9%	1%
W2 L1	.22%	3.15%	5.8%	7.4%
W3 L1	2.4%	15.9%	21.6%	24.0%
W3 L2	3.6%	13.2%	15.3%	15.7%
W3 L3	.40%	.51%	.52%	.52%

Table 1: Measured values in Super Mario Bros.

One interesting aspect to notice in these results is that, in fact, levels tend to have higher difficulty values as the game evolves (less probability of success). It is also possible to notice that different resilience values for the user (T) influences the final probability of success. In particular, a value of 1 represents the less resilient player that can exist, which is one that resigns automatically after failing one simple jump.

This particular game, as some others of the genre, organises the levels in groups named as worlds (in table 1, W stands for world and L for level). One can also notice that when a new world starts difficulty tends to have a small decrease before rising again. This might represent an intention of having a resting level after completing a difficult task, in this case, the last level of the previous world.

Moreover, tuning the exponential coefficient allows adjusting the values in proportions to have them to be more

reliable for usage as real success probabilities. Lower values are more suitable to represent the more skilled players, which are very unlikely to lose in small challenges. Contrary to this, higher values represent less skilled players that have higher failure rates immediately as challenges appear.

In Table 2 we fixed the value of T to 3 and vary the exponential coefficient. So, each column represents the probabilities for players with different skills in the levels presented in each row.

Level	Kd=.25	Kd=.33	Kd=.50
W1L1	40.8%	28.9%	12.6%
W1L2	13.7%	6.8%	1.4%
W1L3	.9%	.2%	.01%
W2 L1	5.8%	2.0%	.2%
W3 L1	21.6%	13.3%	3.3%
W3 L2	15.3%	8.2%	2.1%
W3 L3	.52%	.10%	.003%

Table 2: Measured values for difficulty in Super Mario Bros. with different Kd Values

We proceed now to some tests we ran with a set of players in the game *Little Big Planet*. This game was particularly useful because it allows users to create their own levels. With this, it was possible to create our own test set. We created a small template level with a few jumps and adapted it to have different versions with distinct gap sizes in order to influence the difficulty. In table 2 we present the predicted and the effective probability of success for each obstacle.

Real P(s)	Pred. P(s) Kd=.3	Pred. P(s) Kd=.2	Pred. P(s) Kd=.1
40,4%	40%	54%	74%
89,3%	48%	62%	78%
91,7%	65%	75%	87%
96,2%	57%	69%	83%
96,2%	57%	69%	83%
96,2%	94%	96%	98%
97,1%	78%	84%	92%
98,0%	74%	82%	90%
98,0%	94%	96%	98%
98,0%	87%	91%	95%
98,0%	89%	92%	96%
98,0%	81%	87%	93%
100,0%	95%	96%	98%

Table 3: Examples of measured difficulty in the levels created for the game *Little Big Planet*

Even though we have not yet extracted correlations to the obtained values, some relations are detectable. In further experiments we intend to apply some categorizations to our users, mapped to different values for Kd, in order to achieve for accurate values.

5. RELATED WORK

5.1 Platform videogames

Platform games have been studied before by a few authors. There are three main relevant works to refer in the context of this article. In fact, those three articles are the result of continuous work inside the same research group.

In the first work, Compton and Mateas [Compton06] studied the structure of platform levels, identifying some construction parameters to configure platforms. Also, they identified that platforms are associated together by patterns that represent the actions that the avatar has to do in order to pass through a certain section in one level, designated as cell. They also identified the main structures of organising cells in one level, which was particularly helpful in the definition of the examples presented in the previous sections of this document. Finally, they stated that a system that automatically generates levels could be improved by measuring difficulty. As a theoretical approach, the authors pointed this analysis as one step of the process. In their proposal, one possible approach for this was to calculate all possible trajectories for each jump from one platform to another. With this, they wanted to be able to calculate the spatial window to a possible successful jump. In some aspects, the concept of error margin previously presented in this article represents a window like this. Also, they wanted to use the referred physical calculation to extract the time window the player has to make corrections to the movement.

Later, Smith et al. [Smith08] defined a framework to analyse platform levels in which concerns to its structure. Some important concepts were formalised, resulting in a conceptual model to define a generic level. The need of classifying difficulty was identified once again but only proposed as a future work. Recalling the simplifications or abstractions previously considered to deal with dynamic objects and enemies, the same approach was considered by these authors. For instance, they defined a platform level as a hierarchy where everything that may cause damage to the player is a generic obstacle. This means that one gap between two platforms is an obstacle. Since in our work we directed our measurements to distance, our abstraction was built in the other way around, by mapping objects into gaps.

The previously presented concepts lead to an effective implementation of a level generator [Smith09]. This implementation starts with a rhythm generator that creates a set of actions to be done by the avatar. Those actions are then used with a physics system in order to generate a valid geometry, which is one that allows the user to replicate the generated actions. A system based on critics analyses the generated levels to avoid over generation and also to establish a quality threshold. However, the geometry that is created is still not analysed with a difficulty perspective. Once again, it was identified the need of measuring difficulty for further implementations. Also, the authors stated that measuring difficulty can be used

inside the critics system. The approach we have presented allows analysing a level with the condition of having the set of actions well identified. Once their proposal is rhythm-based, which means that all the level is generated based on a set of actions, difficulty could be measured as we proposed.

Besides the three previous works, there is one particular study for platform games that is important to refer, developed by Pederson et al. [Pedersen09]. Their focus was directed to more abstract concepts about user experience. In particular, the authors studied the users' perceptions in an adapted version of the game *Super Mario Bros*. Several users played the game with different features to establish correlations between those features and reported emotions. A model was built with the obtained results in order to predict fun, challenge and frustration.

5.2 Difficulty measurements in other genres

In other genres, difficulty has been used in some more concretized manners. Just to give a brief overview of that, we will point a few examples.

Togelius et al. [Togelius07] presented a system to automatically generate racing tracks for a driving simulator. To evaluate each track quality for usage as fitness function in a genetic algorithm, they used artificial drivers, mapped to a certain profile, to extract some attributes such as timings and speeds.

A similar approach was used for *Pac-Man-like* games by Togelius and Schmidhuber [Togelius08]. The interesting aspect of this work is that the generation process produces game variants instead of game levels. Again, the main principle used was evolutionary computation and the measurements of difficulty were accomplished by making intelligent agents play the game.

Finally, another interesting work to refer was developed by Pereira et al. [Pereira09]. In this case the objectives and approaches are considerably different from the last ones. The considered genre was Strategic Multiplayer Browser Game, which consists on a slow paced evolution system, accessed by players a few times on one day to establish some strategies about virtual resource management. The system tries to involve the player in an ambient that fits the user preferences, with a balanced distribution of resources to avoid repetition. Also, the authors refer the importance in handling efficiently the constant appearance of new players in the earlier stages of the game.

Naturally, there are several other examples of dynamic adjustment of difficulty, in particular on commercial games. Nevertheless, the presented cases are particularly interesting because they have emphasis on automated generation of content.

5.3 Practical usage of difficulty measures

As presented in this document, the main importance of quantifying difficulty is that it also allows adapting it. Adapting difficulty in videogames has the main objective of levelling a challenge among intervenients. However, it is important to make those adaptations without compromising the core player experience [Hunicke05]. For a single player game, adapting difficulty normally consists on tuning artificial intelligence behind virtual agents. In multiplayer games the key idea is to make the challenge higher to expert players and lower to casual player. The final goal is that every player has a similar chance of winning. Basically, it is the same intent as in some sports that use handicap, such as golf. However, that system is not always applicable, and in some videogames we can make performance analysis in real-time and with that adjust difficulty in a more transparent way. One example of implementation of this concept is the work proposed by Martínez and Mata [Martínez09] in which a *Pong* version is adapted according to players' skills. The authors considered that, in this game, difficulty is due to the paddle size and ball speed. This two attributes are adapted during game play to balance the result and level the odds of each player. In addition, these values are also adapted to make the game challenging enough for both players, and not too easy or hard for both.

6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

We have proposed a way to measure difficulty in platform games and presented its possible application in real games. With basic probability principles, we define the flow of action in a level in association with failing probabilities to obtain a final probability of completing a level with success. Resilience parameters were also considered to reflect the possibility of a user to resign on harder challenges. In addition, basic projectile principles allowed a simple estimation of difficulty for each individual obstacle, which is mapped in a probability value to be used in the calculations of the full success probability.

Our primal experiences show that estimating difficulty based on probability gives an effective notion about its growth. We have tested our measurement in some existing games, in particular a few classics in the gaming industry. As presented previously in section 4, with the results obtained in *Super Mario Bros*., we could verify that our approach generally identifies the earlier levels as being easier than the later ones. This goes in line with the empirical notion of common users and basic design rules.

One important aspect is that the obtained results don't consider the existence of lives (attempts) and checkpoints. So, the probability values presented estimate the chance that the user has to complete the whole levels in one single try.

Finally, our tests with users also allowed identifying a relationship between the proposed difficulty estimator and the real probability of success. Once again, it is noticeable that the difficulty measured can estimate if a level is easier or harder, but it does not represent an effective probability of success in one level.

6.2 Future Work

Even though we have identified some of the principles where difficulty in platform games is based on, some of those need to be analysed with more depth.

Firstly, we believe that it is important to expand the concept of obstacle. The simplifications used for enemies

and dynamics were a sufficient solution for this first approach, but this is a question that requires other calculations to be defined with more detail.

As stated before, the measurements didn't consider the existence of checkpoints and lives that allow the user to retry the level. It would be interesting to expand the study taking this aspect in account and identifying possible differences.

Also, it would be interesting to consider the existence in some levels of the so called *bosses*, which are particularly skilled enemies. For that reason, some levels were not analysed.

Moreover, it is important to make parameter adjustment based in real measurements rather than the ad-hoc tuning that was used. In particular, the coefficient used can represent the user skills, hence predicting more accurately the user's probability of success.

Finally, an interesting way to improve data gathering and analysis, in particular on what concerns to difficulty, might consist in the creation of a community where players could freely play automatically generated levels. This could improve the results obtained in-game.

7. REFERENCES

[Compton06] Compton, K. and Mateas, M. 2006. Procedural Level Design for Platform Games. *Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference* (Stanford, CA, 2006).

[Csikszentmihaly91] Csikszentmihaly, M. Flow: The Psychology of Optimal Experience. Harper Collins, NY, 1991.

[Desurvire04] Desurvire, H., Caplan, M., and Toth, J. A. 2004. Using heuristics to evaluate the playability of games. *CHI '04 Extended Abstracts on Human Factors in Computing Systems* (Vienna, Austria, April 24 - 29, 2004). CHI '04. ACM, New York, NY, 1509-1512.
<http://doi.acm.org/10.1145/985921.986102>

[Hunicke05] Hunicke, R. 2005. The case for dynamic difficulty adjustment in games. *Proceedings of the 2005 ACM SIGCHI international Conference on Advances in Computer Entertainment Technology* (Valencia, Spain, June 15 - 17, 2005). ACE '05, vol. 265. ACM, New York, NY, 429-433.
<http://doi.acm.org/10.1145/1178477.1178573>

[Martínez09] Ibáñez-Martínez, J. and Delgado-Mata, C. From competitive to social two-player videogames. *Proceedings of the 2nd Workshop on Child, Computer and interaction* (Cambridge, Massachusetts, November 05 - 05, 2009). WOCCI '09. ACM, New York, NY, 1-5.
<http://doi.acm.org/10.1145/1640377.1640395>

[Pedersen09] Pedersen, C., Togelius, J., and Yannakakis, G. N. 2009. Modeling player experience in super mario bros. In *Proceedings of the 5th international Conference on Computational intelligence and Games* (Milano, Italy, September 07 - 10, 2009). IEEE Press, Piscataway, NJ, 132-139.

[Pereira09] Pereira, G., Santos, P. A., and Prada, R. 2009. Self-adapting dynamically generated maps for turn-based strategic multiplayer browser games. *Proceedings of the international Conference on Advances in Computer Enterntainment Technology* (Athens, Greece, October 29 - 31, 2009). ACE '09, vol. 422. ACM, New York, NY, 353-356.
<http://doi.acm.org/10.1145/1690388.1690457>

[Smith08] Smith, G., Cha, M., and Whitehead, J. 2008. A framework for analysis of 2D platformer levels. *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games* (Los Angeles, California, August 09 - 10, 2008). Sandbox '08. ACM, New York, NY, 75-80.
<http://doi.acm.org/10.1145/1401843.1401858>

[Smith09] Smith, G., Treanor, M., Whitehead, J., and Mateas, M. 2009. Rhythm-based level generation for 2D platformers. *Proceedings of the 4th international Conference on Foundations of Digital Games* (Orlando, Florida, April 26 - 30, 2009). FDG '09. ACM, New York, NY, 175-182.
<http://doi.acm.org/10.1145/1536513.1536548>

[Togelius07] Togelius, J. Nardi, R and Lucas, S. Towards automatic personalised content creation for racing games, *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007.
<http://cogprints.org/5573/>

[Togelius08] Togelius, J. and Schmidhuber, J. An experiment in automatic game design. *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.141.4110>