Combining Clustering and Grammars for Adaptive Level Generation

Jorge Diz Pico, David Camacho

***Abstract*—public abstract voi– ha ha you funny guy i kill you last**

I. Introduction

Specially in the first years of the industry, disk space was severely limited. A game's length was primarilly constrained by its size in bytes. In the 1980s, when home computing started to rise [CITE], the main shipping mediums were floppy disks (1.44MB) or casettes (X MB), and home computers had main hard drives in the range of X MB. That put a cap on the amount of different playing levels a game could contain. But the design effort also played a big part. A game that wanted to ship with many different levels needed each of them carefully crafted and balanced by the gamemakers, taking a big chunk of the budget because of the time and money involved. The game industry was still in its infancy, with many of the titles being developed by only one person [CITE], and couldn't afford all that investment.

To solve this problem, early games started to develop algorithms to create levels in a procedural way. That is, creating a set of rules to build game levels on the fly, with a random component, ensuring giving birth to procedural content generation. The most classic example if Rogue (COMPANY, YEAR), a roleplaying adventure game where the dungeons are generated at the beginning of each game. The influence of its style spawned a new genre of games called roguelike [CITE].

Even if the disk space limitations are no longer present today, PCG techniques offered other clear advantages that made it be still in use today. The main feature that make procedural algorithms quite appealing for voth developers and players is its guarantee of infinite replayability. A game with a fixed array of levels to play has also a fixed (although possibly quite long) life length. It can be feasible to reach the end of its available possibilities. A game that offers a new, unique level everytime it is run, is virtually unfinishable. Developers can create more experiences with not a bigger effort, and players get longer enjoyment for the same price. Examples can be found in all genres, from adventures (Diablo, developed by Blizzard, 1996), strategy (Civilization, Sid Meier?, BLAH) or exploration (Minecraft, Mojang, 2009).

Having rules in place to create levels can also help prepare the game for unexpected situations. For example, in a strategy game, it could be interesting to explore the dynamics of having less resources on the map. A player may even enjoy this increase in the effective power of the military in securing these resources. But most groundbreaking was the realization that since PCG defines rules for level generation, the parameters controlling the process could be tuned to anticipate the needs of the player. Procedural generation, then, opened the door for a new field: adaptive techniques; that is, the player defining and adjusting the game, indirectly by its behaviour.

For example, the popular Brain Training (developed by, BLAH) configures its memory and reasoning exercises based on a sample test level the players take when first running the game. Another remarkable case is that of Left 4 Dead 2 (developed by Valve, 2009); its "AI Director" module not only creates the level based on previous player data, but also reconfigures the set up *on the fly* if it find the player being low on health and ammunition and unable to face the upcoming challen

In this paper, we propose an architecture for adaptive level generation and test it on the framework of the Mario AI Championship. This initiative is a series of artificial intelligence challenges based on a platform mimicking the popular *Super Mario Bros* saga. One of them, the Level Generation track, has garnered several interesting proposals in the competitions held in previous years at conferences around the world.

In the 2010 competition [CITE], six entries were presented, ranging from genetic algorithms (C) to multi-pass random generators (A). (E), from (X), took an interesting approach. They used heuristics to classify players in three levels of difficulty (low, medium, and hard) and flag the existence of traits from three styles (speed-run, enemy-kill, discovery). Then, this classification set the frequency parameters that were be checked when randomly choosing the next element as the level was constructed left to right.

The architecture presented in this paper is related to that idea, albeit with some new modifications. The classification, for instance, is not a process based on heuristics, but data-driven from records of players. After an assignation has been made, this results in the selection of a set of rules; they affect not only the frequency of elements as in the above model, but also their placement adjacent to each other. This is done to enforce level aesthetics as envisioned by the designer and improve the sense of order.

Jorge Diz and David Camacho are with the Computer Science Department, Escuela Politécnica Superior, Universidad Autónoma de Madrid. c/ Francisco Tomás y Valiente 11, 28049 Madrid, Spain. (email: jorge.diz@estudiante.uam.es and david.camacho@uam.es)

II. Overview

The proposed architecture can be seen in Figure X. It is composed of three layers:

FIGURE X HERE

- **Profiling**: identifies the user and sets a series of parameters to define his profile
- **Derivation**: uses the parameters obtained in the first layer to create a plan to build the level
- **Execution**: the third executes that plan to produce the actual level as output

The three layers are briefly introduced below, as a first approach to their content. Please refer to their own sections, where they are expanded upon, for further information.

A. Profiling

In the first layer, the system ascertains the characteristics of the player. The conclusions reached here will guide the rest of the modules down the correct path. For this reason, it was decided to employ a clustering process. Clustering means that players will be defined by the parameters than make them stand out from the rest, that is, the features that make them special. This allows to focus on those characteristics to tailor the experience to their tastes. A player enjoying a particular part of a stage may or may not be a remarkable event for customization, depending on how often players find that part likable.

Once a player has been assigned to a cluster, the parameters for that cluster are passed down to the second layer.

B. Derivation

The second layer employs the parameters output in the first layer to make a "plan" for building the level. Different players will be assigned different plans, and furthermore, the same player shouldn't be assigned the same plan twice. The chosen method will have to have some inherent randomess.

Levels in platform games can be seen as a succession of repeated basic symbols. For example, in the *Mario* series, some of those symbols might be a gap to jump over, a pipe with an enemy, or a row of blocks. Moreover, the symbols follow some logic in the placement, based on what comes before. Two pipes are never placed immediately adjacent after another, for instance.

Since it became apparent that levels had their own language, it was decided to express their structure as a grammar. The derivation rules act as possible branches to take when building the level, and the symbols of the grammar, the level pieces (called "chunks") to place on it. Each player cluster from the top layer is linked to a grammar, called "schematic", expressing what chunks can be put together and how often in that playstyle.

The schematic is converted to an automaton and traversed, picking the transitions by the weight that their corresponding rule has in the grammar. A list of nodes (a "trace") is output by this process.

C. Execution

Finally the trace is interpreted by the execution layer. Each item on the list produces a call to the game API. This layer converts the plan made by the automaton into an actual playable level, by relying on the level building functions of framework. It's the final intermediary between the architecture proposed here and the game.

In the next sections we'll explore the different layers more in detail, then at the end we'll review the results obtained upon their application.

III. Profiling

This first step of the system is intended to "get to know" the user to control the rest of the process in the right direction. The goals for this layer are to differentiate the player by finding his characteristic playstyle, without the need of asking him, since sometimes, user input can be misleading. For example, when polling the players on their proficiency, newcomer players are known to badly estimate their skills [CITE HERE].

So it was decided to use records for a clustering process. This would classify the players on how they fare in the playstyle spectrum compared to others, and the system can focus on the differential aspects for their personalization. The Mario AI Championship platform stores two kinds of user records: global metrics from each game run, and a detailed log of the actions taken. Since the effort is to evaluate the player on his overall style instead of analyzing specific timestamped events, it was decided to use only the aggregated metrics. Figure Z shows the different parameters the platform records, classified by their category. There are parameters on time, enemy kills, deaths, coins and blocks.

FIGURE Z HERE

Since the clusters are going to be defined by different playing styles, the records have to be collected from presenting the players with comparable challenges. Two players playing one a hard level and another an easy one will produce great disparity in their records, even if their playstyle is quite similar. It was chosen to use the default random levels provided by the Mario AI Championship platform. They are quite simple levels, approachable for beginners but enjoyable by experts. The records from it were thought to be revealing of the players' style (see Results section for the outcome).

The clustering was performed trying to find three specific clusters, identified already in previous works on the field [CITEME]: (in bold, the nicknames given to the them in this paper)

- **speeders**, focused in the main goal (traversing the level) and caring very little for other aspects
- **explorers**, playing the game in no hurry, interacting with every item and walking down every path available
- and **intermediates**, that hold a position in between

These profiles can be thought as a spectrum ranging from quick rush to slow walk, from simple play to extensive interaction. Equivalents can be drawn to similar spectrums in other games. For example, in the card game *Magic: the*

Gathering, the two main strategies for winning are *aggro*, consisting in quickly overwhelming the opponents with simple creatures; and *control*, that slowly builds dominion over the game until dealing the final blow. Other archetypes like *midrange* or *aggro-control* lie somewhere in the middle [CITE ME]. [1]

To find those three clusters, it was decided to rely on the Weka library for the process. It is a powerful, proved library that offers a wide range of clustering algorithms. As a metric to compare them, the value of their log likelihood was chosen, which is a popular paramter to this effect. Log likelihood, however, can't be calculated for non-density-based algorithms, so X, Y and Z had to be wrapped in Weka's MakeDensityBasedCluster to provide this value.

Finally, this process has the advantage of being able to be run offline, that is, previous to the actual level generation. Once the clusters are found in the sampled data, the results can be stored and accessed just when a new player comes. It is then when they are read, the player assigned to one of those clusters and the assignation passed down to the next layer.

IV. Grammars

whew that was long
ok so now to fsms
when we assign the player to a cluster, we implicitly assign him a grammar
since each cluster has a grammar associated
the grammars are called schematics and describe how to build the level
in a pseudo-bnf way
take a look at this simple example
HERE BE FIGURE
the derivation rules have weights
a heavier rule has more probabilities of being chosen
the schematics are context-free grammars
and expands on the right
so in the example we start the level
and we can have a pipe or a coin or flat
and we loop
(ok explain this a bit longer)
the schematic is designed to be infinite
to be able to generate levels of any length
this is a sample of a trace
BLA PIPE COIN RICK ROLL
the schematics are done by hand
the terminals are predefined
they are the available "chunks" that the system can build
for example, a small gap
or a pipe
or a couple of blocks
in this case we made every chunk of the same length, two
we wanted the smallest possible to make it more flexible
the arching structure can be obtained with the rules of the schematic
forcing pieces to be together
but if we wanted to control little details, we need little chunks
but we couldn't do one, because pipes for example are two blocks
and gaps of length one are not very mario-like
we needed chunks of the same size so we had a calculable relation
between chunks used and level length
for the number of iterations
we want this length, ok so we iterate this n of times
also, levels of same length, same number of states
so we can compare them
the choice of cluster from the step before
determines what schematic we read
each one has derivations and weights tailored to their needs
but, as we said, same nonterminals
alright so we read the schematic to construct an automaton representing these transitions
represented by a directed graph where the edges are weighted by their probabilities of being traversed
we use the parse2 library
we defined a grammar for the schematics
then parse2 reads the schematics and transform the rules into transitions
with their weights of course
the resulting graph
can be seen also as a tree with cycles
see the figure for the tree for our example from before
HERE
in this tree model, we'd be traversing it depth-first
we store the terminals we follow, forming a string of chunks to place
we call this string a "genotype", because it represents the level
two equal genotypes represent the same level
an executor later traverses the chunks
from starting node
choses a random transition
pushes the derivation chain into the stack
then pop top node
if it's terminal, we add it to genotype
if it's nonterminal, we pick a random transition and push it to stack
always proceed with the leftmost element of the chain first
because we build the level from left to right

V. Results

god damn that was a lot to tell
so what happened
we used social networks, like twitter, facebook and reddit, and word of mouth
to get the data
117 entries
we are going to try to infer the different existing styles

[1] A third common strategy, combo, exploits peculiarities in the interaction between some elements of the game to win the match instantly. This is, however, a unique strategy exclusive to *Magic*.

so we'll take one sample per person, for these reasons
each person was given a package with ready-to-play game
they were asked to just play, with no other worries about completing the stage, or training before hand
and send back the results
we fed into weka
but we filtered some parameters
because the default stage
has no gaps, or enemies other than goombas
so soe parameters were always zero
those parameters:
we used those, though:
then we tried the five clustering algorithms with default settings
except three clusters as explained
(why only the five? er...)
you can see the results of the log likelihood here
em rocked
the three clusters found represented the profiles we hoped
check this sweet table
isn't it gorgeous
plus they are well separated
check this graph
you like it huh?
speeders like this
explorers like that
and my milkshake brings all the boys to the yard
so we made some grammars
here, take a ride in my intermediate
here's the diagram
and a sample trace for good measure
ok a screenshot too
let me explain some design choices
like, this branch here

VI. Execution

Remember to talk about the two-block chunks. yeah, again.

VII. Conclusions

the separation of layers means that the way you identify users
is not thoroughly linked to how you use that information to guide the building
grammars meant
that we used the same basic chunks
in different ways each
tailoring for styles
grammars are easily intuitive and modifiable
you can control what goes next to what
and how often
and offer choices, different "paths" for the build
we also produced a way of identifying and storing a level
by its genotype of chunks
(did you define chunk? go and do it!)

Future work

we hope
to evaluate this system
with some group control
also
genotypes allow us to use genetic techniques
for example
if the clustering layers defines a player
instad of one membership
with shared membership
we can generate levels from different automata
and mix them
we can also alternate picking derivation rules between grammars by those same percentages

Acknowledgments

This work has been supported by the Spanish Ministry of Science and Innovation under grant TIN2010-19872 (ABANT).

References

[1] T. Roden and I. Parberry, "From artistry to automation: A structured methodology for procedural content creation," in *Proceedings of the 3rd International Conference on Entertainment Computing*, 2004, pp. 151–156.

[2] A. Tychsen, S. Tosca, and T. Brolund, "Personalizing the player experience in mmorpgs," in *Technologies for Interactive Digital Storytelling and Entertainment*, 2006, vol. 4326, pp. 253–264.

[3] M. Booth, "The AI Systems of Left 4 Dead," in *Artificial Intelligence and Interactive Digital Entertainment Conference (Stanford University)*, 2009.

[4] N. Shaker, J. Togelius, G. N. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi, G. Smith, and R. Baumgarten, "The 2010 Mario AI Championship: Level Generation Track," *In special Issue of IEEE Transactions on Procedural Content Generation*, 2010.

[5] G. Smith, M. Treanor, J. Whitehead, and M. Mateas, "Rhythm-Based Level Generation for 2D Platformers," in *International Conference on Foundations of Digital Games*, 2009.

[6] N. Shaker, G. Yannakakis, and J. Togelius, "Towards automatic personalized content generation for platform games," in *Proceedings of Artificial Intelligence and Interactive Digital Entertainment*, 2010.