Combining Clustering and FSM models for Adaptive Level Generation

Jorge Diz Pico, David Camacho

***Abstract*—Content generation has been widely used in gaming to offer virtually infinite replayability. Its combination with adaptive techniques also provides a tighter fit between results and gameplay. A new approach for procedural level generation is presented based on a combination of clustering and finite state machines. The method has been designed and deployed as an entry for the Level Generation track of the Mario AI Championship. The implemented architecture is shown to provide great flexibility for both refinement and extension to other games, and the experimental results are shown to prove the validity of this approach.**

I. Introduction

Procedural level generation, in one way or another, has been present since the inception of games history. Initially, its role was to make up for lack of storage memory for levels by creating them on the fly. It soon revealed its potential to make a game "infinite" by presenting the player with a fresh envinroment every time, a quality best exemplified in early 1980s adventure classic *Rogue* (Toy and Wichman). Since then, it has been featured prominently in many successfully commercial games such as the roleplaying series *Diablo* (Blizzard, 1996) or the exploration-based *Minecraft* (Mojang, 2009).

Lately, some of these techniques have been coupled with adaptive algorithms to make the content generated not only random but also tied to the characteristics of the player. One of the most remarkable is the AI Director found in *Left 4 Dead 2* (Valve, 2009), that even goes as far as to adapt the challenges while the level is played if it finds the player low on health or ammunition [CITE L4D2 PAPER].

In this work, a novel approach to adaptive level generation is presented after testing it on the platform provided by the Mario AI Championship. Previous instances of this competition have featured many worthy techniques [CITE 2010 PAPER]; inspired by them, an architecture was devised that can prove not only equally valid but quite flexible for expansion and adaptation as well.

II. Methodology

The presented approach tries to offer a new insight into adaptively generating random levels. It's comprised of two main parts: a data-driven clustering process (mostly offline), and a finite state machine execution (online). A third layer, a library of functions that actually build the level, serve as the link between the adaptive generator system and the game API.

Jorge Diz and David Camacho are with the Autonomous University of Madrid (email: jorge.diz@estudiante.uam.es and david.camacho@uam.es)

[DIAGRAM OF LAYERS?]

Clustering

The topmost layer undertakes a clustering process on data collected from user play. This approach bases the decisions on actual metrics and avoids overguessing. This execution can be run offline once and then stored for use afterwards, reducing the computational load on the level generation process. It only needs to be rerun when new samples are collected, to refine and recheck the clustering.

The suggested method for collecting samples is to present a vast array of different players with roughly similar levels (ideally, the same). Having them face comparable challenges makes possible to extrapolate on their gameplay style. So as to not overtrain on a particular player, it's recommended to keep just one significant sample per player. Those collections are then clustered as mentioned before.

Now, when a new player comes to the game, he's made to play a test level in the same vein as the ones used in the aforementioned process. The records of his gameplay are fed to a clusterer that will assign him to one of groups found.

It must be noted that the classification shown here is the most basic approach. Clustering could be further refined, perhaps selecting different parameters at different executions, obtaining the player's position along two different axis; or supporting it with additional inputs like the own player's preferred difficulty.

A. Finite state machines

Once the player has been identified as belonging to a given group, this association is used to further assign him an automaton. This is done on a one to an basis: each group has a finite state machine modeled and tailored to its detected needs, defined upon inspection of their characteristic metrics.

This machines have each their own states and transitions, but most importantly, those transitions are defined as probabilistic. Traversing an automaton is done by picking between the next available states from the current one, weighted by the probability associated to that movement. This feature adds the random component needed to keep the levels generated fresh and unique every time, even though the automata themselves are always the same.

This approach allows for a great flexibility on the process given to each user, as different preferences and goals are accounted for. Reducing the space of all possible players into a small number of aggregated playstyles also tones down the number of needed automata as to be few enough to be designed by hand. Manually crafting the states and

weights of the transitions improves the overall arching flow, per the control over sequentially adjacent "chunks" of the level. Albeit keeping it characteristic randomness, the levels are ensured to have an orderly sense instead of being just a messy sequence of pieces.

To further enforce modularity, the automata sit on top of a builder layer that acts as an API through its several level constructing functions. Each state delegates its world-building function to this layer, preventing code repetition on the states and promoting code sharing and reuse.

III. Results

Work was started by collecting data from many different players. Each participant was asked to play one of the levels generated by the standard, random algorithm found in the Mario AI Challenge platform, and return the resulting records. Social networks and word of mouth were used to recruit volunteers, managing to get 115 different user profiles at the time of this article. Several clustering algorithms were then tried to detect the different playstyles. The best results were obtained with Expectation-Maximization, with KMeans and XMeans close behind.

[Some screenshots of Weka comparing how the clusters classify the data, and proving EM is the best fit, would be cool]

Three clear clusters were found. One of the clusters was identifiable by the high number of blocks broken and enemies killed, as well as long completion times. The users in this group were nicknamed "explorers". Another cluster was characteristic for its quick completion time, long use of running mode, and disregard for powerups. They were nicknamed "'speeders". Finally, the remaining cluster, presenting a more neutral set of attributes, were nicknamed "ntermediates".

Per this observation, and reflecting on the meaning of those user profiles, three automata were designed, attending to the inferred needs of those clusters. For example, speeders may frown upon blocks of "stairs" because they often force a stop of the running flow. Similarly, explorers tend to enjoy double rows of blocks, an aspect overlooked by the other types of players.

[A diagram of one of the automata, and some explanation of the logic behind the states and their links would be nice.]

[A panorama with three different screenshots, one from each automata, spanning both columns, would look good here.]

IV. Conclusions

Records belonging to the three different clusters were fed to the algorithm, and playtest showed that each of the levels generated had the capacity of feeling different to the others.

The architecture shown in this paper offers great expansion potential. Its modularity expressed through several layers diminishes the interferences of the different parts. For instance, new player types may be detected in the clustering layer, and the changes in the automata layer would minimal – just adding a new FSM that doesn't break existing functionality. Similarly, changes in an existing FSM wouldn't affect neither the clustering layer nor the other FSMs of the system. Furthermore, new states and transitions may be added without the need for modifications on the underlying library of builder functions they call upon.

Finally, the modelling of the automata after a Markov chain allows for statistical considerations on the probability each state is reached when the automaton is traversed, and opens the possibility for reducing frequencies in a controlled manner.

Future work

The presented architecture can be improved upon in several ways, of which here is suggested the most promising one.

Users are never perfectly encompassed by the cluster they are assigned to; rather, they are more accurately represented by percentages of membership. Rather than force fit a user into a predefined finite state machine, it could be possible to mix and match several of automata until a faithful representation of the user is reached. This could be done by encoding the automata as grammars and using evolutionary algorithms. We hope to explore this path in the near future.

Acknowledgments

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