

# Adaptive Mario : CS8803 Game AI Project 3

Woody Folsom and Marshall Gillson

March 18, 2012

## Introduction

The authors of Adaptive Mario use several PCG techniques to develop a simple platform-game engine which adjusts to player preferences and skills to guide the player toward the ideal flow-state. This architecture should ensure that the maximum amount of replayability is delivered, considering the limited multimedia assets and gameplay mechanics of the Infinite Mario engine on which it is based.

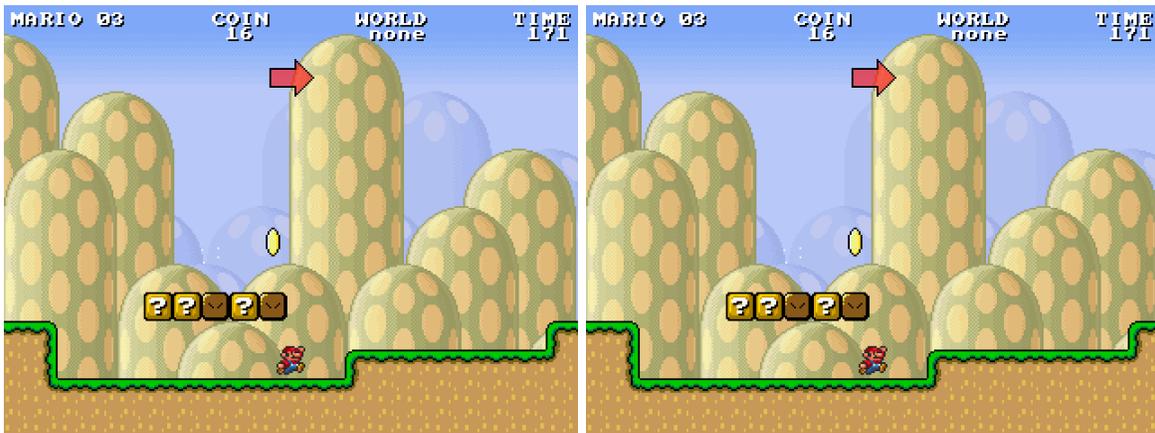


Figure 1: Adaptive Mario in action

As shown in Figure 1, the conceptual goal is that players with different play styles and skill levels should have a very different gameplay experience from the moment the level begins. Of course, the very first time Adaptive Mario is run in a new environment, a specific level is presented based on an a priori model of an 'average' player.

## Related Works

The essential goal of Adaptive Mario is to facilitate flow state by giving the player sufficient challenge and variety without rapidly becoming too difficult. As defined by Mihly Cskszentmihlyi, "flow" is the state of total immersion and cocentration in which the player believes he or she is overcoming obstacles by the narrowest of margins. As shown in Figure 2, achieving this state involves a delicate balance between the difficulty of the game and the player's degree of skill. If the game is too hard, the player will become frustrated. On the other hand, most players will become bored if the game is too easy.

In order to gauge the requisite level is difficulty, it is first necessary to develop a model of the player's ability. Numerous models of player psychology have been proposted since Richard Bartle's 1996 study of MUD (Multi-User Dungeon) players [1]. Some examples are:

**Richard Bartle's Player Types** Killer, Socializer, Achiever, Explorer

**Nick Yee's Player Motivations** Relationships, Immersion, Grief, Achievement, Leadership

**John Radoff's Player Motivations** Immersion, Cooperation, Achievement, Competition

The Adaptive Mario PlayerProfile assigns the player one of five roles, based on past performance: BUMPER, COLLECTOR, JUMPER, RUNNER and SHOOTER. This model is loosely based on Bartle's extended model, with BUMPER corresponding to manipulator, in the sense that the player is observed to manipulate the environment by bumping bricks and throwing shells. This model differs from the examples above primary in that social elements are not evaluated, since Mario is a single-player game.

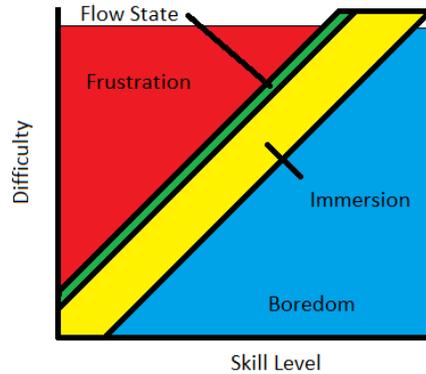


Figure 2: Flow State: Game Difficulty vs. Player Skill

## Level Generator Design

The Adaptive Mario level generator functions as a 4-stage pipeline composed of a Profile Matcher, Level Archetype Selector, Level Generator and Challenge Component Creator.

### Profile Matcher

The first step of the level generation process is to evaluate the player’s style and skill level. Adaptive Mario uses the output file “player.txt” generated by the Gameplay class during the most recent game, if available. In addition, more detailed statistics from the DataRecorder class are preserved as XML and are used by some components of the level generator. The ProfileMatcher scores aspects of gameplay such as number of kills, jumps and coins collected in order to evaluate the player’s skill in the areas of running, jumping, collecting, shooting. This vector of skills, rated on a range of 1 - 100, is then compared against a precalculated ‘typical’ vector for each of the five player types using a simple mean-squared error metric. The player is then given the profile matching the most similar set of skills.

In addition to assigning a player profile, the ProfileMatcher assesses the player’s skill level by comparing specific metric against an ideal score assigning the player one of five skill levels: Novice, Beginner, Competent, Proficient or Expert. These categories are based on the Dreyfus model of skill acquisition with each successive level representing an achievement of 20

Score	Skill Level	Attributes
20%	Novice	low situational awareness, reflexive responses
40%	Beginner	uses judgement to react to challenges, limited awareness
60%	Competent	copies with multiple challenges, uses sound strategy
80%	Proficient	makes rapid decisions, prioritizes goals
100%	Expert	intuitively solves challenges, pushes boundaries

### Level Archetype Selection

The Level Archetype selector picks an overall macro-form for the level based on the player’s profile and skill level. As there are five categories and only three distinct environments in the Infinite Mario engine (Overland, Underground and Castle), different profiles cause similar Level Archetypes to be generated. However, this does provide the opportunity to indicate to the player what types of challenges lie ahead and may serve to enhance immersion.

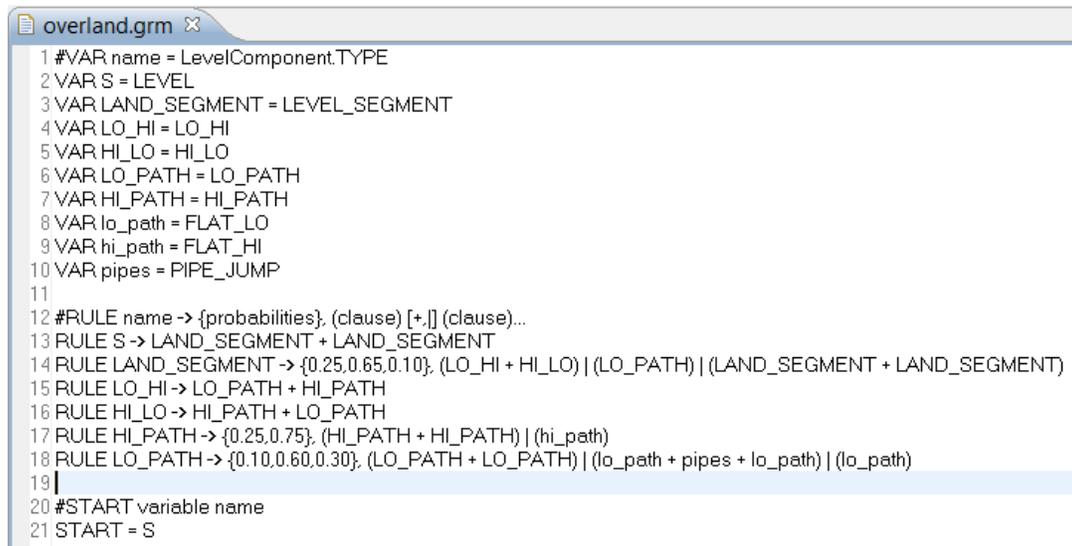
Next, the Level Archetype is further customized for the player by disabling challenges and enabling rewards according to a predned set of rules. In order to allow the rule set to be easily customized and extended, Adaptive Mario uses Drools, an implementation of Charles Forgy’s 1982 Rete algorithm. Once the appropriate rules have fired, level generation proceeds to the third stage: macro-structure generation.

### Level Generation: Macro-structure

The task of the macro-structure generator is to deliver a list of sequential Mario level design elements (LevelComponents) based on a high level description similar to “Overland level with gaps but no pipes, difficulty 5/10”. Several algorithms were considered during the planning phase of this module. A Genetic Algorithm approach could potentially produce novel levels, but judging the relative fitness of candidate levels could be problematic. Hierarchical task networks (HTNs) and

rhythm-based approaches have also been used, but tend to generate levels with a very uniform structure in a single pass. For ease of modification and to leverage the authors' design intuition, a context-free grammar (CFG) was implemented instead.

Given a suitable array of predefined LevelComponent types, each of which corresponds to one or more Infinite Mario game sprites, the structure of a level archetype can be completely specified in the form of a file-based CFG. As shown in the following example, the OR clauses are assigned probability values, giving a stochastic property to the generated level structure.



```
overland.grm x
1 #VAR name = LevelComponent.TYPE
2 VAR S = LEVEL
3 VAR LAND_SEGMENT = LEVEL_SEGMENT
4 VAR LO_HI = LO_HI
5 VAR HI_LO = HI_LO
6 VAR LO_PATH = LO_PATH
7 VAR HI_PATH = HI_PATH
8 VAR lo_path = FLAT_LO
9 VAR hi_path = FLAT_HI
10 VAR pipes = PIPE_JUMP
11
12 #RULE name -> {probabilities}. (clause) [+.] (clause)...
13 RULE S -> LAND_SEGMENT + LAND_SEGMENT
14 RULE LAND_SEGMENT -> {0.25,0.65,0.10}. (LO_HI + HI_LO) | (LO_PATH) | (LAND_SEGMENT + LAND_SEGMENT)
15 RULE LO_HI -> LO_PATH + HI_PATH
16 RULE HI_LO -> HI_PATH + LO_PATH
17 RULE HI_PATH -> {0.25,0.75}. (HI_PATH + HI_PATH) | (hi_path)
18 RULE LO_PATH -> {0.10,0.60,0.30}. (LO_PATH + LO_PATH) | (lo_path + pipes + lo_path) | (lo_path)
19 |
20 #START variable name
21 START = S
```

Figure 3: A Simple Overland Level Grammar

One potential pitfall of using a CFG for this purpose is that the generated level may be over-specified (containing too many elements) and hence too crowded, or under-specified and nearly empty. To mitigate this problem, a fitness evaluation function iteratively invokes the LevelGrammar, rejecting proposed levels with too many or too few LevelComponents.

## Challenge Components: Micro-structure

### Evaluation

### Conclusion

It seems clear from the results that the framework of Adaptive Mario has the potential to guide the player toward his or her own idealized version of a platform game, while still presenting a reasonable level of challenge. Not only does difficulty scale in proportion to the player's performance (preventing frustration), but care is taken in the design of the level grammar to avoid repeatedly giving the player 'more of the same' (leading to boredom). One potential issue exists, however - it is possible that some players simply are not fans of the platform jumper genre!

Fortunately, the architecture of Adaptive Mario (and the externalization of the rule engine and level grammar in particular) means that the designer is not limited to the side-scrolling platform jumper palette. It is easy to imagine that Adaptive Mario could be transformed into a Roguelike or adventure game without substantially altering the core Mario mechanics. Just as the player is freed from the constraints of pregenerated content, the designer (or hobbyist) is given the necessary tools to alter the game world as desired.

## Appendix A: Building the Game

Building the Project 3 executable requires a Java SDK version 1.6+ and Apache Ant.

To build the game, execute `ant clean` followed by `ant` from the main project directory.

## Appendix B: Running the Game

To run the game, change to the 'dist' subdirectory following a successful build and execute `java -jar CS8803.P3.jar`.

`seed` random number generator seed value (long)

## References

- [1] Richard Bartle. Hearts, Clubs, Diamonds, Spades: Players Who suit MUDs. <http://aigamedev.com/open/interviews/mario-ai/>, 1996. [Online; accessed 18-March-2012].