Utilizing an AI Approach to Pathfinding in Super Mario

Nick De Tullio, Michael Jalkio, & Thomas Gautier
nrd24@cornell.edu, mrj77@cornell.edu, tng26@cornell.edu

Abstract

This paper utilizes the Mario AI benchmark, to describe the success of both previously implemented Mario bots compared to our own implementation. The Mario AI Benchmark was introduced for use first in the IEEE Games Innovation Conference (ICE-GIC) and then later in Computational Intelligence Games (CIG) as a benchmark for reinforcement learning algorithms and other game AI techniques in a public domain clone of the platform game *Super Mario Bros*. Each year the CIG conducts a Mario AI Competition, in which they compare contestants submitted Mario bots to this benchmark. The goal of this paper is to determine the success of bots that were already created for this competition and then to decide whether an improved bot can be created. The bots that will be evaluated fell into three categories in the competitions A*-based, learning-based, and rule-based implementations.

Introduction

A platform game, or platformer, is a video game which involves guiding an avatar to jump between suspended platforms, over obstacles, or both in order to advance the game. These challenges are known as jumping puzzles or freerunning. It is the job of the player to control the jumps in order to avoid letting their avatar fall from the platforms or miss the necessary jumps. The most common element of the genre of platform games is the jump button. Platform games originated in the early 1980s in side scrolling or 2D video games, which were eventually followed by 3D successors in the mid-1990s.

Super Mario Bros is a series of platform video games created by Nintendo, featuring the main character Mario who serves as the player's avatar. The game follows Mario's adventures in the fictional Mushroom Kingdom, where he runs and jumps across platforms and atop enemies in themed levels. In the game it is necessary for Mario to jump over enemies and between platforms. It is also beneficial to Mario to jump on enemies in order to prevent them from coming across his path again, provided that they do not have spiked turtle shells. Aside from staying alive and progressing towards the finish line, it is also beneficial to the player's score for Mario to collect coins by jumping into them as they are found in the level, as well as by jumping into bricks for coins that may be hidden. There are also a multitude of power-ups and items which give Mario special powers such as fireball-throwing, size-changing, and extra lives, which can also be found by jumping into bricks.

In order to make the benchmark possible the game was modified via the construction of an API that enabled it to be easily interfaced with learning algorithms and the various competitors' controllers. The modifications included the removal of the dependency on the system clock so that the learning algorithm can "step" forward, removing the dependency on the graphical output, and substantial refactoring. Each step in the game corresponds to 40 milliseconds of simulated time which has an update frequency of 25 frames per second. At each step, the controller receives a description of the environment, and outputs an action. The software that makes these modifications possible is a single threaded Java application, which allows

for the key methods that a controller needs to implement to be specified in a single Java interface file.

There are many features that make *Super Mario Bros* or platformers in general particularly interesting from an artificial intelligence or reinforcement learning perspective. What is likely the most important feature of the game is its potentially very rich and high-dimensional environment representation. When a human player views the game, he views a small part of the current level in two dimensions, with the screen centered on Mario. There are very rarely sparse views, in most cases there are dozens of objects such as brick blocks, enemies and collectable items. The static environment, such as grass, pipes, and brick blocks are laid out in a grid that covers approximately $19 * 19$ cells.

The entries for the 2009 Mario AI competition were classified into three broad categories, hard-coded heuristic, learning-based, and A*-based controllers. Hard-coded heuristics were the largest category, comprised of seven different controllers which were hand-constructed, non-adaptive, and did not use search-based methods for action selection. Learning based controllers were the second largest category comprised of two subcategories artificial evolution. There were three entries, the first used genetic programming, the second evolved code for a stack-based virtual machine, and the third evolved a rule-based controller.

Controller	D0	D3	D5	D10	Score	Total Kills	Mario Status	Time Left	Mario Mode	Total Sum
FJ	10660.2	2170.2	1994.2	1137.4	15962.0	71	9	6611	32	22685.0
FJ2	8186.2	1859.1	1605.3	1158.7	12809.3	60	8	6470	31	19378.3
GP	11108.5	2404.5	1397.9	1024.2	15935.2	99	10	6009	46	22099.2
RB	11369.9	6555.2	3130.2	3291.9	24347.2	228	12	4968	19	29574.2
A*	11628.8	11611.2	11651.2	11619.2	46510.4	414	40	4730	80	51774.4
Amazing	11267.4	3103.8	2742.0	1199.3	18312.6	83	10	6035	21	24461.6

Table 1: Controller Scores

Results of our run of the various bots and the

The Environment and State Space

As stated above, the environment was based on Markus Persson's *Infinite Mario Bros* which is an online, public domain clone of Nintendo's classic *Super Mario Bros* with open-source Java code. *Infinite Mario Bros* was primarily designed to be human-playable as an entertaining game, so many of the features of the source were incompatible or inconvenient for AI integration. Therefore, Karakovskiy et al. edited the *Infinite Mario Bros* source to include an API that allowed interfacing with AI code, removing dependency on graphics output and the system clock to allow simulations to be run computationally and not graphically, with a temporal step size in simulated time of 40ms (25 Hz). The source was also extensively modified to include a more sophisticated level-generation class which allowed for both more complex levels than were available before and the ability to recreate levels using each level's seed. Karakovskiy et al. also added a significant number of parameters for level design that allowed programmers to specify certain aspects of each level, such as an enemy mask which determined the presence of various types of enemies, difficulty, length, and the frequency of dead ends that require backtracking (dead ends were not originally included in *Infinite Mario Bros*). Their final software resulted in a cross-platform Java program that ran on a single thread and had low computation times for standard desktop machines of the time (they quote an estimate of order 10 levels per second on a 2009 MacBook with most of the computation time being used by the agents and not by the game simulation software).

The API consisted of two main interfaces and a "Task" analysis interface. The benchmark software provides the environment interface to the AI which includes all the information available to it in-game. The environment is classified in a $22 * 22$ "block" grid, with each block taking up a significant amount of pixel space. In each block, there were binary fields for passible/impassible terrain and for the presence of enemies. Because block coordinates were far too inaccurate for sufficient path calculations, the designers implemented exact (pixel space) coordinates for enemies in order to allow AI consoles the accuracy of human perception. Finally, the environment interface contained information about Mario's current state, including his mode (small, big, or fire), if he is on the ground, if he can jump, and if he is carrying a Koopa shell.