The Mario AI Benchmark and Competitions

Sergey Karakovskiy and Julian Togelius

***Abstract*—This paper describes the Mario AI benchmark, a game-based benchmark for reinforcement learning algorithms and game AI techniques developed by the authors. The benchmark is based on a public domain clone of Nintendo's classic platform game *Super Mario Bros*, and completely open source. During the last two years, the benchmark has been used in a number of competitions associated with international conferences, and researchers and students from around the world have contributed diverse solutions to try to beat the benchmark. The paper summarises these contributions, gives an overview of the state-of-art in Mario-playing AIs, and chronicles the development of the benchmark. This paper is intended as the definitive point of reference for those using the benchmark for research or teaching.**

Keywords: game AI, reinforcement learning, benchmarking, competitions

I. Introduction

When doing research in computational and/or artificial intelligence applied to games, it is important to have suitable games to apply the AI algorithms to. This applies regardless of whether one is doing research on using games to test and improve artificial intelligence (games provide challenging yet scalable problems which engage many central aspects of human cognitive capacity), or whether one is doing research on using CI/AI methods to improve games (for example with player satisfaction modelling, procedural content generation and creation of believable and interesting bots). No single game will ever satisfy all projects and directions within this steadily growing research field, as different games pose different challenges. However, the community has much to gain from standardising on a relatively small set of games, which are freely available and on which competing CI/AI methods can be easily and fairly compared.

A "perfect benchmark game" would have to satisfy numerous criteria. It should test a number of interesting cognitive abilities, preferably such that are not effectively tested by other benchmark games already out there. It should be "easy to learn but hard to master", in other words either have a tunable challenge level or have a naturally deep learning curve, so that it differentiates between players and algorithms of different skill at all levels. It should be visually appealing, easy to understand and generally something that spectators know and care about. People should like to play it. The policy representation (or input/output space) should be sufficiently general that a number of different CI/AI methods can be applied to it without too much work.

SK is with St. Petersburg State University, Universitetskii prospekt 35, Petergof, 198504 Saint-Petersburg, Russia. JT is with the Center for Computer Games Research, IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen, Denmark. Emails: sergey@idsia.ch, julian@togelius.com

The technical platform is also important. The benchmark game implementation should run on major computing platforms available now and in the foreseeable future, and should run identically on all systems. The software needs to be simple to install, the API easy to understand and it should be possible for anyone with adequate programming knowledge to have a simple solution up and running within five minutes, otherwise many researchers will choose to use their own benchmarks which they know better. The implementation needs to be computationally lightweight, and able to be sped up to many times (hundreds or thousands of times) real-time performance. This last criterion is particularly important for applying learning algorithms to the game.

In this paper we present the *Mario AI Benchmark*, a benchmark software based on *Infinite Mario Bros* which in turn is a public domain clone of Nintendo's classic platform game *Super Mario Bros*. We argue that this benchmark satisfies all of the criteria laid out above to at least some extent, and therefore is highly suitable for several kinds of CI/AI research. We also describe the competitions that have been held during 2009 and 2010 based on successive versions of the Mario AI Benchmark. These competitions have attracted a reasonably large number of submissions and considerable media attention, and as a result the benchmark is now used in a number of university courses worldwide.

The structure of the paper is as follows. First, we discuss other competitions and benchmarks, and the characteristics of the game this particular benchmark is based on. We then describe the benchmark, including the API and the *AmiCo* and *Punctual Judge* libraries which permits the benchmark to be used efficiently and fairly from diverse programming languages. This is followed by a description of how the competitions based on the benchmarks were organised and the results of the individual competitions. In connection with this we discuss how the evolution of both the benchmark and the competition entries was informed by the advances in playing capability displayed by the best entries in each competition. A final section discusses other research that has used the Mario AI Benchmark, how you can use the benchmark in your own research and teaching and what we can learn from the competitions. The concluding acknowledgements make clear how this paper differs from other papers that have been published previously about this benchmark and these competitions.

A. Previous game-based competitions and benchmarks

Chess is probably the oldest known artificial intelligence benchmark, and has played an important role in CI/AI research since Turing first suggested that the game could be automatically played using the MiniMax algorithm [1]. In the

famous *Kasparov vs. Deep Blue* match in 1997, a computer program for the first time beat the human grandmaster and became the world's best chess player [2]. The exact significance of this event is debated, but what was proven beyond doubt was that an AI implementation can excel at a particular game without necessarily having a broad behavioural repertoire or being able to adapt to a variety of real-world challenges. The related board game Checkers (draughts), which was used for influential early machine learning experiments [3] has recently been completely solved using tree-search methods and can now be played perfectly (perfect play by both players leads to a draw) [4]. Another game where it is no longer interesting to try to beat humans is *Scrabble*; the best Scrabble-playing programs (such as *Maven*) can win over all humans without searching more than one turn ahead, because of the advantage of quick and complete dictionary access [5].

Other board games have delivered harder challenges for CI and AI; international competitions have been set up where competitors can submit their best game-playing programs and play against each other for a number of board games. In recent years, much work has focused on the ancient oriental game of *Go*. The high branching factor of this game makes traditional tree-search techniques ineffective, and the winners of recent Go competitions have been based on Monte Carlo techniques [6]. Other turn-based, originally non-digital games games such as the board game *Backgammon* and the card game *Poker* feature non-determinism or incomplete information, which seems to necessitate statistical models to play well. In such games, the current best computer programs are typically not yet competitive with human grandmasters, as evidenced by competitions where humans and programs can play each other.

While board games and card games certainly pose many hard and interesting challenges and, in humans, require cognitive abilities such as reasoning and planning to play well, there are many relevant challenges that are not posed by such games. Digital games, in particular video games, might require planning and reasoning to play well; but they may also require such capacities as visual pattern recognition, spatial navigation and reasoning, prediction of environmental dynamics, short-term memory, quick reactions, very limited information and ability to handle continuous multi-dimensional state and action descriptions. Additionally, video games are visually and culturally more appealing than board games for many people – especially young people, a fact which can help draw students into CI/AI research and advertise the field to the general public.

With this in mind, during the last decade a number of competitions have been organized in conjunctions with international conferences, several of them sponsored by the IEEE Computational Intelligence Society. Some of these competitions are based on arcade-style games such as *Ms. Pac-Man* [7], *Cellz* [8], X-pilot [9] and a simple 2D racing game [10]. But there are also competitions based on the first-person shooter *Unreal Tournament* [11], [12], the modern racing game *TORCS* [13] and the real-time strategy game *StarCraft* [14]. The organization, results and summaries of entries of many of these competitions have been written up as journal articles or conference papers, providing an archival reference point for other researchers wishing to use the benchmarks developed for their own experiments; for other competitions, at least the benchmark software and initial experiments have been published.

B. Platform games as an AI challenge

Platform games can be defined as games where the player controls a character/avatar, usually with humanoid form, in an environment characterised by differences in altitude between surfaces ("platforms") interspersed by holes/gaps. The character can typically move horizontally (walk) and jump, and sometimes perform other actions as well; the game world features gravity, meaning that it is seldom straightforward to negotiate large gaps or altitude differences.

To our best knowledge, there have not been any previous competitions focusing on platform game AI. The only published papers on AI for platform games we know of is a recent paper of our own where we described experiments in evolving neural network controllers for the same game as was used in the competition using an earlier version of the API [15], and our earlier conference paper on the first iteration of this competition. Some other papers have described uses of AI techniques for automatic generation of levels for platform games [16], [17], [18]; some of this research was done using versions of the Mario AI Benchmark [19], [20], [21].

Most commercial platform games incorporate little or no AI. The main reason for this is probably that most platform games are not adversarial; a single player controls a single character who makes its way through a sequence of levels, with his success dependent only on the player's skill. The obstacles that have to be overcome typically revolve around the environment (gaps to be jumped over, items to be found etc) and NPC enemies; however, in most platform games these enemies move according to preset patterns or simple homing behaviours. (This can be contrasted to other popular genres such as first-person shooters and real-time strategy games, where the single-player modes require relatively complex AI to provide an entertaining adversary for the player.)

Though apparently an under-studied topic, artificial intelligence for controlling the player character in platform games is interesting from several perspectives. From a game development perspective, it would be valuable to be able to automatically create controllers that play in the style of particular human players. This could be used both to guide players when they get stuck (cf. Nintendo's recent "Demo Play" feature, introduced to cope with the increasingly diverse demographic distribution of players) and to automatically test new game levels and features as part of an algorithm to automatically tune or create content for a platform game.

From an AI and reinforcement learning perspective, platform games represent interesting challenges as they have high-dimensional state and observation spaces and relatively high-dimensional action spaces, and require the execution of different skills in sequence. Further, they can be made into good testbeds as they can typically be executed much faster than real time and tuned to different difficulty levels. We will go into more detail on this in the next section, where we describe the specific platform game used in this competition.

C. Infinite Mario Bros

The Mario AI benchmark is based on Markus Persson's *Infinite Mario Bros*, which is a public domain clone of Nintendo's classic platform game *Super Mario Bros*. The original Infinite Mario Bros is playable on the web, where Java source code is also available[1].

The gameplay in Super Mario Bros consists in moving the player-controlled character, Mario, through two-dimensional levels, which are viewed sideways. Mario can walk and run to the right and left, jump, and (depending on which state he is in) shoot fireballs. Gravity acts on Mario, making it necessary to jump over holes to get past them. Mario can be in one of three states: *Small*, *Big* (can crush some objects by jumping into them from below), and *Fire* (can shoot fireballs). Getting hurt by an enemy means changing to previous mode or dying. While the main goal is to get to the end of the level, auxiliary goals include gaining a high score by collecting items and killing enemies, and clearing the level as fast as possible.

1) Automatic level generation: While implementing most features of Super Mario Bros, the standout feature of Infinite Mario Bros is the automatic generation of levels. Every time a new game is started, levels are randomly generated by traversing a fixed width and adding features (such as blocks, gaps and opponents) according to certain heuristics. The level generation can be parameterised, including the desired difficulty of the level, which affects the number and placement of holes, enemies and obstacles. The original Infinite Mario Bros level generator is somewhat limited; for example, it cannot produce levels that include dead ends, which would require back-tracking to get out of, and does not allow for specifying random seeds that allow the recreation of particular levels. For the Mario AI Benchmark we have enhanced the level generator considerably, as will be detailed below.

2) The challenges of playing Infinite Mario Bros: Several features make Super/Infinite Mario Bros particularly interesting from an AI or reinforcement learning perspective. The most important of these is the potentially very rich and high-dimensional environment representation. When a human player plays the game, he views a small part of the current level from the side, with the screen centred on Mario. Still, this view often includes dozens of objects such as brick blocks, enemies and collectable items. The static environment (grass, pipes, brick blocks etc.) and the coins are laid out in a grid (of which the standard screen covers approximately $19 * 19$ cells), whereas moving items (most enemies, as well as the mushroom power-ups) move continuously at pixel resolution.

The action space, while discrete, is also rather large. In the original Nintendo game, the player controls Mario with a D-pad (up, down, right, left) and two buttons (A, B). The A button initiates a jump (the height of the jump is determined partly by how long it is pressed); the B button initiates running mode and, if Mario is in the Fire state, shoots a fireball. Disregarding the unused up direction, this means that the information to be supplied by the controller at each time step is five bits, yielding $2^5 = 32$ possible actions, though some of these are nonsensical (e.g. left together with right).

Another interesting feature is that there is a smooth learning curve between levels, both in terms of which behaviours are necessary and their required degree of refinement. For example, to complete a very simple Mario level (with no enemies and only small and few holes and obstacles) it might be enough to keep walking right and jumping whenever there is something (hole or obstacle) immediately in front of Mario. A controller that does this should be easy to learn. To complete the same level while collecting as many as possible of the coins present on the same level likely demands some planning skills, such as smashing a power-up block to retrieve a mushroom that makes Mario Big so that he can retrieve the coins hidden behind a brick block, and jumping up on a platform to collect the coins there and then going back to collect the coins hidden under it. More advanced levels, including most of those in the original Super Mario Bros game, require a varied behaviour repertoire just to complete. These levels might include concentrations of enemies of different kinds which can only be passed by timing Mario's passage precisely; arrangements of holes and platforms that require complicated sequences of jumps to pass; dead ends that require backtracking; and so on. How to complete Super Mario Bros in minimal time while collecting the highest score is still the subject of intense competition among human players[2].

II. The benchmark

In order to build a benchmark out of Infinite Mario Bros, we modified the game rather heavily and constructed an API that would enable it to be easily interfaced to learning algorithms and competitors' controllers. The modifications included removing the dependency on the system clock so that it can be "stepped" forward by the learning algorithm, removing the dependency on graphical output, and substantial refactoring (Markus Persson did not anticipate that the game would be turned into an RL benchmark). Each time step, which corresponds to 40 milliseconds of simulated time (an update frequency of 25 fps), the controller receives a description of the environment, and outputs an action. The resulting software is a single-threaded Java application that can easily be run on any major hardware architecture and

[1] http://www.mojang.com/notch/mario/

[2] Search for "super mario speedrun" on YouTube to gauge the interest in this subject.

operating system, with the key methods that a controller needs to implement specified in a single Java interface file. On a MacBook from 2009, $10-40$ full levels can be played per second (several thousand times faster than real-time); for anything but trivial agents, most of the computation time is spent in the agent rather than in the benchmark.

A. API

The application programming interface (API) of the Mario AI benchmark can be broken down into the following Java interfaces:

1) The Environment interface: Describes the game state to the agent at each time step. The main types of information presented are:

- One or several *receptive field* observations. These are two-dimensional arrays that describe the world around Mario with block resolution, and with Mario himself in the center. In the first version of the benchmark, one receptive field contained binary information about the environment (where 0=passable terrain and 1=impassable, such as blocks and platforms) and another receptive field contained binary information about the enemies on screen (1 for enemies, otherwise). In later versions, the ability for agents to change *Z-level*, the level of detail for objects in the receptive field informations, was added. Initially, all receptive fields had the dimensions 22×22, but in later versions this became a property of the level. Figure 1 illustrates a small receptive field around Mario, used in early neural network experiments.
- *Exact positions* of enemies. As the receptive field observations have block resolution, they might not provide enough detail for some agents. Therefore, a list of x and y positions relative to Mario with pixel resolution is provided in later versions of the benchmark.
- *Mario state*. Information about what state Mario is in (Small, Big, Fire), whether Mario is currently on the ground, can currently jump and is currently carrying the shell of a Koopa (turtle-like enemy) is provided as separate binary/discrete variables.

Additionally, the possibility of receiving the raw bitmap of the rendered game screen was implemented, but has not been used in competitions so far.

2) The Agent interface: This is the only interface that needs to be implemented in order to create a functional Mario-playing agent. The key method here is *getAction*, which takes an *Environment* as input and returns a five-bit array specifying the action to take. The original Super Mario Bros game is controlled by the Nintendo controller featuring a four-dimensional directional pad (d-pad) and two buttons, A and B; when played by a human, a similar arrangements of keys on the keyboard is used for Infinite Mario Bros. The five bits correspond to pressing or not pressing each of the two buttons A and B, and three of the four directions (left, right and down) - the up direction has no meaning with the feature set we are implementing. Left and right moves Mario left right, down makes Mario duck, A initiates a jump and

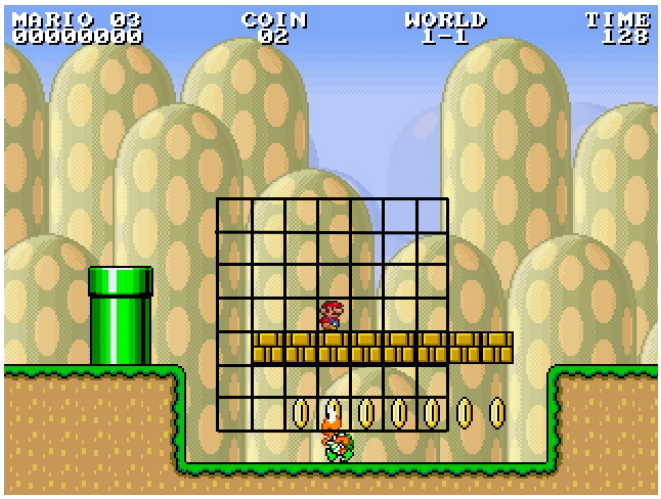

Fig. 1. A small receptive field around Mario. Each grid cell is the size of one block.

B makes Mario run if pressed together with left or right and additionally makes Mario shoot a fireball if in Fire mode. All of these buttons can be pressed simultaneously. This yields a total of $2^5 = 32$ actions, though several of these are pointless and not commonly used (e.g. pressing left and right simultaneously).

3) The Task interface: The Task defines certain aspects of the gameplay, including the presence or absence of sensory noise, whether there should be intermediate rewards and exactly which evaluation function is used.

B. Tunable level generator

The initial versions of the benchmark used the standard level generator that comes with Infinite Mario Bros, though slightly changed to allow for the specification of random seeds. As competition entries became more sophisticated, it became evident that the existing level generator could not provide levels of sufficient diversity and challenge. A new level generator was therefore constructed, which can construct harder, more diverse and (in the opinion of the authors) more interesting levels. Figure 2 illustrates the workflow of the level generator. The basic idea is to add "zones" from left to right until the required length has been reached, where both the content of the zones and their placement can be modified in multiple ways depending on the parameters given to the generator. In contrast to the original level generator, the new generator has more than 20 tunable parameters. The following parameters are among the most important:

- *Seed:* any level can be recreated by specifying the exact same parameters, including random number seed.
- *Difficulty:* affects the complexity of levels generated, size of gaps etc.
- *Type:* overground, underground, castle (indoor environment used for boss fights).
- *Length:* the length of the level. The *time limit* can also be controlled.

- *Creatures:* bit mask specifying the presence/absence of particular creatures.
- *Dead ends:* the frequency of level constructs that may force the player to backtrack and try another route.
- *Gravity:* affects how high and far Mario can jump. Several other physical properties (e.g. friction, wind) can also be controlled.

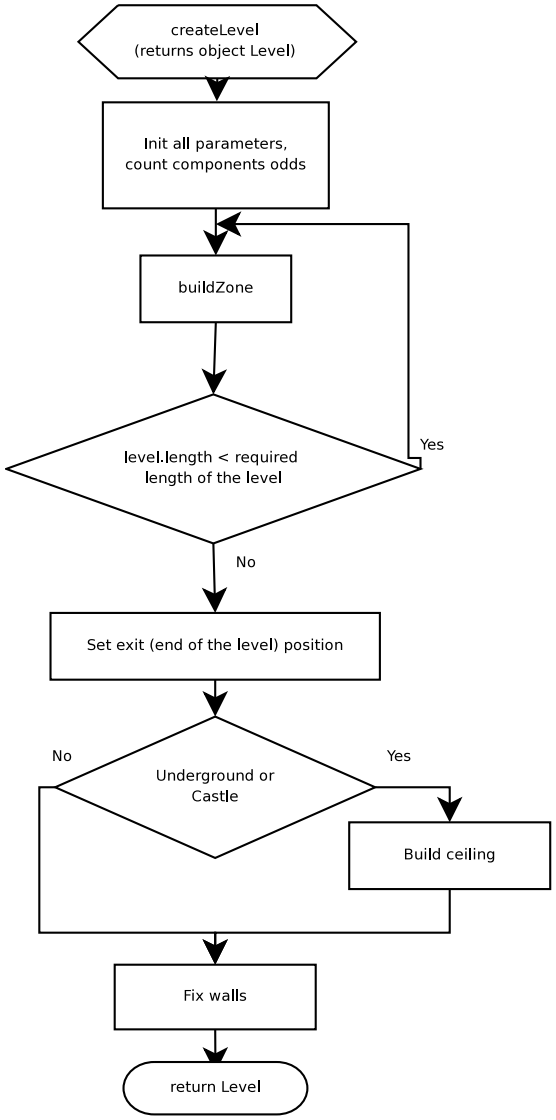

Fig. 2. Workflow for the tunable level generator.

Figure 3 depicts part of a level generated by the tunable level generator.

C. The AmiCo library

In the first versions of the benchmark, a TCP interface for controllers was provided so that controllers written in other languages than Java could be interfaced to the code. However, this TCP interface introduced a considerable communication overhead making non-Java agents orders of magnitude slower, and had occasional stability issues. For later releases, a new library called *AmiCo* was developed for communication between the benchmark and agents developed in other language.

The AmiCo library is applicable for inter-language process communication beyond the Mario AI Benchmark. The purpose of the library is to provide an easy-to-use and as seamless as possible bridge between foreign programming languages preserving high performance, comparable to the native languages' runtime speeds. The idea behind it is to make use of the native language bindings of various languages, such as *JNI* for Java, *ctypes* for Python and *HSFFIG* for Haskell. Currently it has bindings for the above mentioned three languages, but can easily be extended to others. This native bridge is possible due to native C++ bindings for both Java and Python. The library allows the programmer to invoke both static and non-static methods of a target `Java` class and provides complete access to JNI[3] methods from Python.

D. Punctual Judge

Punctual Judge is the part of the benchmark software that is responsible for fair timing of controllers across computers and changing computer load. Like AmiCo, this part of the software can readily be used outside the Mario AI benchmark, for example in other time-critical benchmarking applications.

When Punctual Judge is activated, a custom classloader loads the user-provided Mario controller, instruments it on the fly through injecting additional byte code and returns an instrumented class, which can be called by the benchmark. During evaluation, Punctual Judge counts the number of byte codes executed. Exceptions are disregarded, as any exception will terminate the benchmarking software.

Experimental runs show that Punctual Judge gives an additional overhead of only about 32% for Java, a factor which could conceivably be optimised further if necessary.

Using Punctual Judge, a competitor can get accurate information about the number of byte code instructions his controller performs before submission. As this number is machine-independent, this information allows the competitors to match the competition time bounds tightly without running the risk of the controller being disqualified because of differing performance profile of the computer on which the scoring is done.

For more information about the level generator, Punctual Judge and AmiCo, please refer to [22].

III. Competition organization

The organization and rules of the competition sought to fulfil the following objectives:

1) *Ease of participation.* We wanted researchers of different kinds and of different levels of experience to be able to participate, students as well as withered professors who haven't written much code in a decade.

[3]Java Native Interface

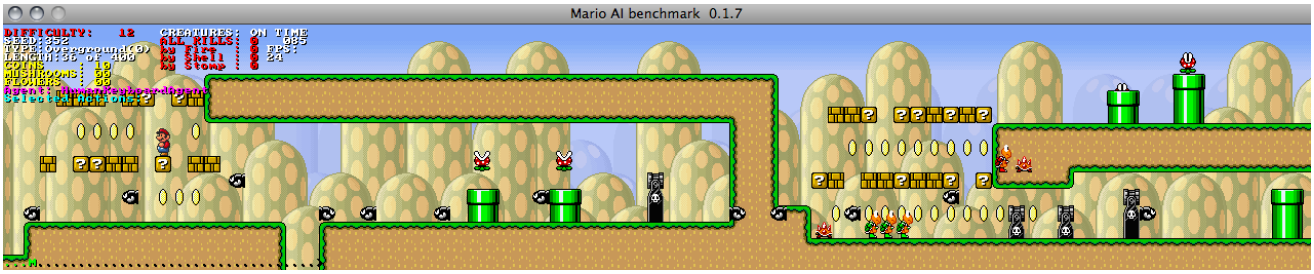

Fig. 3. Part of a level generated by the tunable level generator. Mario can be seen standing standing on a question mark near the left end of the picture. Note that this elongated screenshot contains the same information as approximately four standard screens. Under normal conditions, it would not be possible to judge from the place Mario is standing whether it would be possible to proceed by walking under or on top of the the overhanging platform adjacent to the right.

2) *Transparency.* We wanted as much as possible to be publicly known about the competing entries, the benchmark software, the organization of the competition etc. This can be seen as an end in itself, but a more transparent competition also makes it easier to detect cheaters, to exploit the scientific results of the competition and to reuse the software developed for it.
3) *Ease of finding a winner.* We wanted it to be unambiguously clear how to rank all submitted entries and who won the competition.
4) *Depth of challenge.* We wanted there to be a clear score difference between controllers of different quality, both at the top and the bottom of the high-score table.

After each iteration of the competition, these four objectives were evaluated, and changes were introduced to the benchmark and competition organization if any objective was not fulfilled. While the two first objectives have generally been well met throughout the competition, several changes have been introduced in order to better meet the two latter objectives, as we will see below.

The competition web page hosts the rules, the downloadable software and the final results of the competition[4]. (For the 2009 edition of the competition, a different webpage was used[5]). Additionally, a Google Group was set up to which all technical and rules questions were to be posted, so that they came to the attention of and could be answered by both organisers and other competitors[6], and where the organisers posted news about the competition. The searchable archive of the discussion group functions as a repository of technical information about the competition.

Competitors were free to submit controllers written in any programming language and using any development methodology, as long as they could interface to an unmodified version of the competition software and control Mario in real time on a standard desktop PC running Mac OS X or Windows XP. For competitors using only Java, there was a standardised submission format. Any submission that didn't follow this format needed to be accompanied by detailed instructions for how to run it. Additionally, the submission needed to be accompanied by the score the competitors had recorded themselves using the benchmark software, so that the organisers could verify that the submission ran as intended on their systems. We also urged each competitor to submit a description of how the controller works as a text file.

Competitors were urged to submit their controllers early, and then re-submit improved versions. This was so that any problems that would have disqualified the controllers could be detected and rectified and the controllers resubmitted. No submissions or resubmissions at all were accepted after the deadline (about a week before each competition event).

IV. The 2009 Competitions

In 2009, two phases of the competition were run. The first was associated with the IEEE Games Innovation Conference (ICE-GIC) conference in London in August, and the second was associated with the IEEE Conference on Computational Intelligence and Games (CIG) in Milan, Italy, in September. The results of each phase were presented as an event during the conference it was associated with.

A. Media campaign

A media campaign was initiated through stories on social media websites *Digg* and *Slashdot*. At about the same time one of the competitors (Robin Baumgarten) posting a video of his controller online. This video quickly went viral, and gathered six hundred thousand views in a few days. This attracted the attention of mainstream and popular science media, resulting in several articles about the competition and research associated with it [23], [24], [25]. We believe that these articles contributed substantially to the number of qualified entrants to the competition, while at the same time dissuading some less advanced and/or ambitious potential competitors from entering.

B. Summary of competition entries

The 15 different entries submitted to the two phases of the 2009 Mario AI competition can be classified into three broad categories.

[4]http://www.marioai.org
[5]http://julian.togelius.com/mariocompetition2009
[6]http://groups.google.com/group/marioai

1) Hand-coded heuristic: This was the largest category. Seven different controllers were submitted which were hand-constructed, non-adaptive and did not use search-based methods for action selection. All of these were very quick to return an action when prompted, implying that a low amount of computation was performed. Trond Ellingsen, Sergio Lopez, Rafael Oliveira and Glenn Hartmann submitted rule-based controllers, that determined the action to return based on verifying a number of relatively simply conditions. Spencer Schumann augmented one of the sample rule-based controllers with a bit of internal simulation to determine the end position of possible jumps. Mario Perez submitted a controller based on the subsumption architecture, common in robotics, and Michal Tulacek built his controller around a finite state machine.

2) Learning-based: Five controllers were based on offline training. Three of these used artificial evolution: Matthew Erickson evolved expression trees of the type commonly used in genetic programming; Douglas Hawkins evolved code for a stack-based virtual machine; and Erek Speed evolved a rule-based controller. Sergey Polikarpov trained a controller based on the "cyberneuron" neural network architecture using a form of ontogenetic reinforcement learning, and Alexandru Paler trained a neural network to play using supervised learning on human playing data.

3) A-based:* The stars of the 2009 competition were the A*-based controllers. These agents reduce the problem of how to safely navigate the levels to the problem of how at any point to get to the rightmost edge of the screen, and cast this problem as a pathfinding problem. The *A* search algorithm* is a widely used best-first graph search algorithm that finds a path with the lowest cost between a pre-defined start node and one out of possibly several goal-nodes [26]. This algorithm is used to search for the best path in *game state space*, which is different from simply searching in the space of Mario's positions and requires that a fairly complete simulation of the game's dynamics is available to the search algorithm. Fortunately, given that the game is open-source and computationally lightweight, it is reasonably simple to copy and adapt the game engine to provide such a simulation.

The first of these controllers was submitted by Robin Baumgarten, who also posted a video showing his agent's progress on a level of intermediate difficulty on YouTube. This video quickly garnered over six hundred thousand views, and gave a considerable boost to the publicity campaign for the competition. (A screenshot of Robin's agent in action, similar to what was depicted in the viral video, is shown in figure 4.) The proficiency of the controller as evident from the video inspired some competitors, while dissuading others from participating in the competition. Before the deadline, two other controllers based on A* had been submitted to the competition, one by Peter Lawford and another by a team consisting of Andy Sloane, Caleb Anderson and Peter Burns. These controllers differed subtly from Robin's controller in both design and performance, but were all among the top entries. More information about

Fig. 4. Visualization of the future paths considered by the Robin Baumgarten's A* controller. Each red line shows a possible future trajectory for Mario, taking the dynamic nature of the world into account.

Robin Baumgarten's controller can be found in [27].

C. Scoring

All entries were scored before the conference through running them on 10 levels of increasing difficulty, and using the total distance travelled on these levels as the score. The scoring procedure was deterministic, as the same random seed was used for all controllers, except in the few cases where the controller was nondeterministic. The scoring method uses a supplied random number seed to generate the levels. Competitors were asked to score their own submissions with seed 0 so that this score could be verified by the organizers, but the seed used for the competition scoring was not generated until after the submission deadline, so that competitors could not optimise their controllers for beating a particular sequence of levels.

For the second phase of the competition (the CIG phase) we discovered some time before the submission deadline that two of the submitted controllers were able to clear all levels for some random seeds. We therefore modified the scoring method so as to make it possible to differentiate better between high-scoring controllers. First of all, we increased the number of levels to 40, and varied the length of the levels stochastically, so that controllers could not be optimised for a fixed level length. In case two controllers still cleared all 40 levels, we defined three tie-breakers: game-time (not clock-time) left at the end of all 40 levels, number of total kills, and mode sum (the sum of all Mario modes at the end of levels, where 0=small, 1=big and 2=fire; a high mode sum indicates that the player has taken little damage). So if two controllers both cleared all levels, that one that took the least time to do so would win, and if both took the same time the most efficient killer would win etc.

D. Results

The results of the ICE-GIC phase are presented in table IV-D, and show that Robin Baumgarten's controller

performed best, very closely followed by Peter Lawford's controller and closely followed by Andy Sloane et al.'s controller. We also include a simple evolved neural network controller and a very simple hard-coded heuristic controller (the *ForwardJumpingAgent* which was included with the competition software and served as inspiration for some of the competitors) for comparison; only the four top controllers outperformed the ForwardJumpingAgent.

Competitor	progress	ms/step
Robin Baumgarten	17264	5.62
Peter Lawford	17261	6.99
Andy Sloane et al.	16219	15.19
Sergio Lopez	12439	0.04
Mario Perez	8952	0.03
Rafael Oliveira	8251	?
Michael Tulacek	6668	0.03
Erek Speed	2896	0.03
Glenn Hartmann	1170	0.06
Evolved neural net	7805	0.04
ForwardJumpingAgent	9361	0.0007

TABLE I
RESULTS OF THE ICE-GIC PHASE OF THE 2009 MARIO AI COMPETITION. THE NUMBERS IN THE "PROGRESS" COLUMN REFER TO HOW FAR THE AGENT GOT TOWARDS THE END OF THE LEVEL, SUMMED OVER ALL LEVELS; "MS/STEP" REFERS TO HOW MANY MILLISECONDS EACH AGENT ON AVERAGE TAKES TO RETURN AN ACTION AFTER PRESENTED WITH AN OBSERVATION.

For the CIG phase, we had changed the scoring procedure as detailed in section IV-C. This turned out to be a wise move, as both Robin Baumgarten's and Peter Lawford's agents managed to finish all of the levels, and Andy Sloane et al.'s came very close. In compliance with our own rules, Robin rather than Peter was declared the winner because of his controller being faster (having more in-game time left at the end of all levels). Peter's controller, however, was better at killing enemies.

The best controller that was not based on A*, that of Trond Ellingsen, scored less than half of the A* agents. The best agent developed using some form of learning or optimisation, that of Matthew Erickson, was even further down the list. This suggests a massive victory of classic AI approaches over CI techniques. (At least as long as one does not care much about computation time; if score is divided by average time taken per time step, ForwardJumpingAgent wins the competition...)

E. Result evaluation and benchmark improvements

The most obvious message of the 2009 competition was the superiority of the A*-based agents over everything else that was submitted. Search in state-space for the fastest way to move right using simulation of the game engine was clearly superior to all reactive approaches. Looking a bit closer at the results, it is clear that the top two A* agents had very similar scores and in the CIG phase of the competition they could only be distinguished based on auxiliary criteria such as the amount of time left at the end of levels or the number or creatures killed. This is because those controllers *cleared every level in the competition*. Either the A* algorithm was the final answer to how to play platform games, or the levels that were part of the competition did not accurately represent the challenges posed by levels in the original Super Mario Bros and other platform games.

We analysed the functioning of the A* algorithm for level features which were present in real Super Mario Bros levels but not in the competition levels, and which would pose problems for the A* algorithm. One feature in particular stood out: dead ends. A dead end is a situation where the player can choose to take at least two different paths forward, but at least one of these paths is blocked, requiring the player to backtrack and choose another path. It is important that it is not possible to see which path is blocked at the time of choosing; this means that the blocked path must be at least half a standard screen long. For the 2010 Mario AI Championship, the level generator was extended to include the possibility of generating dead ends. (Figure 3 shows a dead end generated by the augmented level generator.) Additionally, a number of other changes were introduced to the level generator to make it possible to create harder levels, such as greater control over numbers of particular items and possibility of hidden blocks and longer gaps. It was decided to increase the difficulty of the hardest levels in future competitions and including some levels which were literally impossible to finish to test the behaviour of controllers on such levels.

None of the winning controllers incorporated any kind of learning. This is not a problem in itself, as the rules stipulated that any kind of controller was welcome and the objective was to find the best AI for platform games regardless of underlying principles. However, a variation the same benchmark could conceivably also be used to test the capabilities of learning algorithms to be integrated into platform game controllers. We therefore decided to broaden the competition by introducing a new track of the competition dedicated to this.

The playing style of the A*-based controllers is very far from human-like. A video of e.g. Robin Baumgarten's controller playing looks very different from a video of a human playing the same level; the controller is constantly running and jumping rightwards, and has a spooky exactness in that it tends to jump off platforms at the very last pixel. Indeed, this machine-like quality of the gameplay is probably a major reason for why the YouTube video of Baumgarten's agent became so popular. While the gameplay not being human-like is not a problem for the competition, the same benchmark could conceivably be used to compete in human-like gameplay as well, and therefore a new competition track was introduced dedicated to this.

Finally, the recent interest in procedural content generation within the game AI community [28], [29] suggested to us that the benchmark could be used as the basis for a content generation competition as well. A new track was therefore devised for 2010, focusing on programs that generate levels.

Competitor	approach	progress	levels	time left	kills	mode
Robin Baumgarten	A*	46564.8	40	4878	373	76
Peter Lawford	A*	46564.8	40	4841	421	69
Andy Sloane	A*	44735.5	38	4822	294	67
Trond Ellingsen	RB	20599.2	11	5510	201	22
Sergio Lopez	RB	18240.3	11	5119	83	17
Spencer Schumann	RB	17010.5	8	6493	99	24
Matthew Erickson	GP	12676.3	7	6017	80	37
Douglas Hawkins	GP	12407.0	8	6190	90	32
Sergey Polikarpov	NN	12203.3	3	6303	67	38
Mario Perez	SM, Lrs	12060.2	4	4497	170	23
Alexandru Paler	NN, A*	7358.9	3	4401	69	43
Michael Tulacek	SM	6571.8	3	5965	52	14
Rafael Oliveira	RB	6314.2	1	6692	36	9
Glenn Hartmann	RB	1060.0	0	1134	8	71
Erek Speed	GA	out of memory				

TABLE II

RESULTS OF THE CIG PHASE OF THE 2009 MARIO AI COMPETITION. EXPLANATION OF THE ACRONYMS IN THE "APPROACH" COLUMN: RB: RULE-BASED, GP: GENETIC PROGRAMMING, NN: NEURAL NETWORK, SM: STATE MACHINE, LRS: LAYERED CONTROLLER, GA: GENETIC ALGORITHM. EXPLANATION OF COLUMN LABELS: PROGRESS, AS IN THE PREVIOUS TABLE; LEVELS: NUMBER OF LEVELS CLEARED (OUT TO 40); TIME LEFT: SUM OF IN-GAME SECONDS LEFT AT THE END OF EACH LEVEL (A HIGHER NUMBER MEANS THAT THE AGENT FINISHED THE LEVEL FASTER); KILLS: NUMBER OF ENEMIES KILLED; MODE: NUMBER OF MODE SWITCHES, MEANING THE NUMBER OF TIMES THE AGENT LOST A MODE (THROUGH GETTING HURT) OR GAINED A MODE (THROUGH COLLECTING A MUSHROOM OR FLOWER).

V. THE 2010 CHAMPIONSHIP

The 2010 Mario AI Championship consisted of four separate tracks:

- The *Gameplay track* was the direct continuation of the 2009 Mario AI Competition. Like in that competition, the goal for submitted controllers was to clear as many levels as possible, and the rules were the same. The main difference to the year before was the incremental addition of new features to the benchmark API, and the more diverse and harder levels used to test the controllers on.
- The *Learning track* was created to test learning agents, or in other words to disadvantage agents that do not incorporate any learning (online of offline). Agents are tested on levels that are unseen during (human) development of the agent, but the agent is allowed to train on the track before being scored. More precisely, each agent was allowed to play each testing track 10000 times, but only the score from 10001st playthrough contributed to the final score. This way, agents that incorporated mechanisms for learning how to play a particular track could do better than those that were overall good players but lack the ability to specialize.
- The *Turing Test track* responds to the perceived machinelike quality of the best controllers from the 2009 track, by asking competitors to submit controllers that behave in a human-like fashion. The controllers were assessed by letting an audience of non-expert humans view a number of videos of humans and agents playing the same level, and for each video voting on whether the player was human or machine.
- The *Level Generation track* used the Mario AI benchmark software for a procedural content generation competition. Competitors submitted personalized level generators, that could produce new, playable Infinite Mario Bros levels given information about the playing style and capabilities of a human player. The level generators were assessed by letting humans play first a test track, and then levels generated on-line specifically for them by two different generators, and choosing which one of the generated levels was most engaging.

Both the Gameplay, Learning and Turing Test track used variations on the same interface, meaning that the same agents could be submitted to all three tracks with minor changes. The Level Generation track, on the other hand, used a radically different interface as the submitted software was asked to do something quite different from playing the game.

The championship was run in association with four international conferences on AI/CI and games. Not every track was run at every competition event:

- *EvoGames, part of EvoStar*; Istanbul, Turkey, 7 April: Gameplay and Learning tracks.
- *IEEE World Congress on Computational Intelligence*; Barcelona, Spain, July: Gameplay track and a dry run for the level generation track.
- *IEEE Conference on Computational Intelligence and Games*; Copenhagen, Denmark, August: Gameplay, Learning and Level generation tracks.
- *IEEE Games Innovation Conference*; Hong Kong, China, 24 December: Turing test track.

In this paper, the organization, competitors and results of the Gameplay and Learning tracks are discussed. As there is simply not room to discuss all four tracks to a satisfactory level of detail within a single journal article, the other two tracks have been described elsewhere. For more about the Level generation track, see [30]; the Turing test track is discussed further in [31].

A. The gameplay track

The 2010 championship saw both new (5) and old (3) competitors entering, and the best controllers were signifi-

cantly better players than previous years. We kept improving the benchmark as described above between the three competition events, and therefore the scores attained in different events are not directly comparable. In particular, the EvoStar competition event did not yet include levels with dead-ends (which were part of the two later competition events) though it did include levels that were overall harder than those in the 2009 competition.

Because of the gradual evolution of the interface, and the fact that most interface changes were additions of new modes of experiencing the environment, almost complete backwards compatibility has been maintained for controllers. This means that participants in the 2009 competition could enter the 2010 gameplay track with none or only minor changes to their controllers. Therefore, a relatively high number of participants has been maintained throughout the 2010 competition events, and new ideas could easily be compared with the best of the previous controllers. In particular, Robin Baumgarten entered all three gameplay events with incrementally refined versions of the controller that won the 2009 competition. Still, there were fewer competitors in 2010 than there were in 2009, which can be explained partly by that we could not get the same media attention as we got for the 2009 competition, and partly by that the levels were harder and several of the competitors more mature, suggesting that newcomers with weaker entries that would have submitted their entries if they thought they had a chance of winning chose not to do so as they thought the competition too stiff.

One strong new contender in the 2010 championship was the *REALM* agent, due to Slawomir Bojarski and Clare Bates Congdon. This agent is built on sets of rules, which are evolved offline to maximize the distance travelled by the agent. An agent is built up of a set of 20 rules, where each rule has a handful of preconditions that test for relatively primitive aspects of the game state, such as whether Mario may jump or there is an enemy above to the left. The consequences of the rules, on the other hand, are relatively high level plans (such as move to the right of the screen or kill the nearest enemy or bypass the dead-end) which are executed with the help of A* planning. More information on this entry can be found in [32].

Another interesting newcomer was the bot by Diego Perez and Miguel Nicolau, which uses grammatical evolution (a form of genetic programming) to evolve behaviour trees. Descriptions of different versions of that agent can be found in [33], [34].

As can be seen from table V-A, Bojarski and Congdon's REALM agent won the CIG event of the 2010 championship, which was the final of that year. The superiority of the REALM approach was evident in that it not only reached the highest overall distance score, but also cleared the most levels, killed the most enemies and was not disqualified even once. The runner up was Sergey Polikarpov's CyberNeuron agent, which won the previous competition event at IEEE WCCI (see table V-A) and also came second in the EvoStar event V-A. Robin Baumgarten's revised controller finished third with a very high number of disqualified levels, meaning that it often timed out when faced with a situation where it could not find a path to the left end of the screen. As the most difficult levels became more difficult between each competition, Robin's agent dropped from first to second to third place.

However, not even the REALM agent managed to clear all levels. In some cases, it got stuck in a particularly vicious dead end, or failed to clear a very long jump. These two types of situations are responsible for the vast majority of deaths and disqualifications for all of the top controllers - it was rare to see any of these controllers lose a life to enemy collisions. Some of the levels in the test contain gaps that cannot be bypassed in a single jump, but only through stomping on a bullet or flying koopa mid-air, an operation that requires good timing and is usually quite hard for a human to execute. All of the top controllers were occasionally able to display such feats, which would seem like the outcome of careful planning to a casual human spectator.

B. The learning track

As described above, the submission format for the learning track was intentionally very similar to that of the gameplay track, and the same agent could with minimal modifications be submitted to both tracks. In terms of evaluation, the difference is that while in the gameplay track each agent is tested once on a number of levels, in the learning track the agent is tested 10001 times on the same level and only the score from the last attempt counts. The challenge is to use the first ten thousand attempts to learn to play this particular level as well as possible.

Three of the four participants in the learning track were variations of controllers submitted to the gameplay track. Slawomir Bojarski and Clare Bates Congdon participated in the learning track with the "full version" of the REALM controllers, having the evolutionary rule learning mechanism turned on and using the ten thousand trials for fitness evaluations [32]. The evolutionary run is seeded with the same set of rules that won the gameplay track.

The FEETSIES Team: (Erek Speed, Stephie Wu and Tom Lieber) submitted an entry where the policy (represented as direct mappings from screen observations to actions) was optimised between trials by "Cuckoo Search via Lévy Flights", a recent biologically inspired stochastic search algorithm [36]. The search was seeded with the policy of the simplistic heuristic ForwardJumpingAgent, in other words to continuously run rightwards and jump. Starting from this policy, the search process identifies the situations where the agent should do something else, via mutations that randomly select another action for a given state. On top of the state-action mapping, hardcoded heuristics deal with searching for hidden blocks and retreating from dead ends. The agent is described in more detail in [35].

Laura Villalobos was the only participant in the learning track that did not submit to the gameplay track. Her solution was based on genetic programming, dividing the 10000 trials

Competitor	score	disqualifications	technique
Robin Baumgarten	634468	3	A*
Sergey Polikarpov	301775	9	CyberNeuron (RL)
Alexander Buck	290204	0	?
Eamon Wong	253867	0	Q-learning
Mathew Erickson	167862	0	Genetic Programming

TABLE III
RESULTS OF THE EVOSTAR EVENT OF THE 2010 MARIO AI CHAMPIONSHIP, IN DESCENDING RANK ORDER. EXPLANATION OF COLUMN LABELS: SCORE: SUMMED SCORE FOR THE AGENT BASED ON PROGRESS, KILLS AND TIME TAKEN, AND USED TO CALCULATE THE WINNER; DISQUALIFICATIONS: NUMBER OF TIMES THE AGENT WAS DISQUALIFIED FOR TAKING TOO LONG TIME TO RETURN AN ACTION AFTER BEING PRESENTED WITH AN OBSERVATION; TECHNIQUE: WHAT THE CONTROLLER WAS BASED ON.

Competitor	score	disqualifications	technique
Sergey Polikarpov	1637935	1	CyberNeuron (RL)
Robin Baumgarten	1537834	312	A*
Robert Reynolds, Leonard Kinnaird-Heether & Tracy Lai	1113437	0	Elman network / cultural algorithm
Alexander Buck	991372	0	?
Eamon Wong	972341	0	Q-learning
Mathew Erickson	634239	0	Genetic Programming

TABLE IV
RESULTS OF THE WCCI EVENT OF THE 2010 MARIO AI CHAMPIONSHIP, IN DESCENDING RANK ORDER.

Competitor	score	disqualifications	levels cleared	kills	reference
Slawomir Bojarski and Clare Bates Congdon	1789109.1	0	94	246	[32]
Sergey Polikarpov	1348465.6	4	82	156	
Robin Baumgarten	1253462.6	271	63	137	[27]
Diego Perez and Miguel Nicolau	1181452.4	0	62	173	[33], [34]
Robert Reynolds and Erek Speed	804635.7	0	16	86	[35]
Alexander Buck	442337.8	0	4	65	
Matthew Erickson	12676.3	7	60	80	
Eamon Wong	438857.6	0	0	27	

TABLE V
RESULTS OF THE CIG EVENT OF THE 2010 MARIO AI CHAMPIONSHIP, IN DESCENDING RANK ORDER. EXPLANATION OF COLUMN LABELS: LEVELS CLEARED: NUMBER OF LEVELS CLEARED; KILLS: NUMBER OF ENEMIES KILLED.

into 25 generations with a population of 400 individuals, using tree-based program representation and a standard set of GP instructions. The terminals (inputs) corresponded to the presence of objects and enemies in the standard grid observation. Meanwhile, Robin Baumgarten submitted the same A* agent as to the gameplay track without any significant changes.

The results of the competition are presented in table VI. The most striking result is that all three agents that incorporate learning between trials perform vastly better than the non-learning agent, even though that agent is one of the better entries for the gameplay track. While the winner of the learning track (Bojarski and Congdon) outperformed the non-learning controller (Baumgarten) in the gameplay track as well, the difference is much larger in the learning track. This shows that the learning controllers were indeed able to benefit from the time given to adapt to particular levels. (In turn, this shows that the design of the learning track was successful in advantaging learning controllers.) Upon visual inspection of the 10001st attempt of any of the learning controllers on any particular level, a number of behaviours are found which indicate having learnt how to play a particular level rather than levels in general. These include jumping in the air to reveal known hidden blocks, and always choosing the right path when presented with two paths, one of which is a dead end.

It is also interesting to note that the two best-performing submissions, despite both relying on stochastic global search in some form, are quite different. Whereas one uses an evolutionary algorithm, the other uses Cuckoo search; one uses a compact rule-based policy representation that maps particular features of the state to actions, whereas the other uses a sparse and direct mapping of complete states to actions; finally, only the second uses hard-coded rules for dead ends.

VI. DISCUSSION

A. Evaluating the competition

In section III we laid out four objectives that we sought to fulfill in the design and running of the competition. These were ease of participation, transparency, ease of finding a winner and depth of challenge.

Ease of participation was mainly achieved through having a simple web page, simple interfaces, simple sample controllers available and letting all competition software be open source. Participation was greatly increased through

Participant	Affiliation	Score
Slawomir Bojarski and Clare Bates Congdon	University of Southern Maine	45017
FEETSIES Team (Erek Speed, Stephie Wu, Tom Lieber)		44801
Laura Villalobos		41024
Robin Baumgarten	Imperial College, London	19054

TABLE VI
LEARNING TRACK RESULTS, CIG 2010 EVENT, COPENHAGEN

the very successful media campaign, built on social media. Transparency was achieved through forcing all submissions to be open source and publishing them on the web site after the end of the competition. However, the short descriptions submitted by competitors have in general not been enough to replicate the agents, or perhaps even to understand them given the source code, and therefore it has been very welcome that several of the competitors (including two competition winners) have published their agent designs as academic conference papers.

The two latter objectives proved to be somewhat more tricky. In the second competition event of 2009, the top two controllers managed to clear all levels and therefore had the same progress score; auxiliary performance measures had to be used in order to find a winner. The addition of harder levels including longer gaps, hidden blocks and dead ends changed this situation, and during the last competition event, no agent was able to clear all levels, and there was significant difference in progress score between the best controller and the runner-up. Therefore, all objectives can currently be seen as fulfilled.

B. AI for platform games

It was a bit disappointing for the organisers (and no doubt some of the competitors) to see the levels in the 2009 competition events yield so easily to the A*-based agents. Would the whole problem of playing platform games be solvable by a four decades old (and rather simple) search algorithm? This seemed improbable, given the grip classic platform games such as Super Mario Bros has held over generations of players, and the skill differentiation among even very experienced players of such games.

The addition of more complex features to the levels for the 2010 competition events showed that this was indeed not the case. In order to handle dead ends, the agent needs to identify when it is stuck, decide to retrace its steps, decide for how long to do this before attempting a new path, and finally remember which path was the wrong one so as not to take it again. It could be argued that this is algorithmically trivial, but the challenge is for the agent to perform these relatively high-level actions integrated with the low-level actions of avoiding enemies, navigating gaps and platforms etc. From a robotics perspective, the challenge could be formulated as that of carrying out plans in a complex environment using an embodied agent, even if the embodiment is within a virtual world. This AI problem seems to call for a hierarchical solution, so it is not surprising that the winner of the 2010 championship (due to Bojarski and Congdon) employs a two-level solution, where rules specify higher-level plans that are executed by a lower-level mechanism.

While there are still advances to be made given the current set of game elements and level generator features and settings, there is scope for increasing and diversifying the challenge further by integrating more level elements from existing platform games (including Super Mario Bros). Some examples are moving platforms, which would require the player to model the system of platforms and await the right moment to start a sequence of jumps, and sequences of switches and doors (or keys and locks), which would require the player to plan in which order to press various buttons (or pick up keys) in order to proceed.

C. Using the Mario AI benchmark for your own research and teaching

The Mario AI Competition web page, complete with the competition software, rules and all submitted controllers, will remain in place for the foreseeable future. We actively encourage use of the rules and software for your own events and courses. The Mario AI benchmark software is used for class projects in a number of AI courses around the world;either for a well-defined exercises or as an environments that students can use for implementing a term project. It is unrealistic to demand that a student produce a controller that competes with the current best approaches during a simple course project — creating a world-class Mario AI player using some interesting technique would rather be suitable as a half-year advanced project, such as a masters thesis.

When organising courses or local competitions using the Mario AI Benchmark, it is worth remembering that the existing Google Group and its archive can serve as a useful technical resource, that the result tables in this paper provide a useful point of reference, and that existing sample controllers help students get started quickly. We appreciate if students are encouraged to submit their agents to the next iteration of the Mario AI Championship.

The benchmark software can also be used as a tool for your own research. In addition to the several papers cited referenced above, which describe various submitted entries to the competition, a number of papers have been published by various authors where the main goal was not to win the Mario AI Championship — the following is a sample:

Handa [37] investigated techniques for reduction of the dimensionality of the input space, so as to make the problem tractable for standard reinforcement learning algorithms. It

was shown that such algorithms could perform well on the problem when the right sort of dimensionality reduction was used. In a similar vein, Ross and Bagnell try to reduce the dimensionality of the input space, but for the purpose of imitation learning [38]. Karakovskiy [22] applied multidimensional recurrent neural networks the problem, and was able to train controllers that played particular levels very well using this novel neural network architecture, and which generalised better to unseen levels than other neural network architectures. Shaker et al. [39] recorded video of players' faces while playing the Mario AI benchmark (controlling the character manually) and used machine learning techniques to predict player behaviour and experience from facial expressions. In addition, a number of authors have attempted to predict player experience from playing style and to generate entertaining/interesting levels automatically, but these publications relate more to the level generation track of the championship [30].

VII. Conclusions

This paper has described the Mario AI Benchmark, and the various competitions that have been held based on it in 2009 and 2010 (except the level generation and Turing test competitions, which are described elsewhere). As the paper does not include competition participants as authors, the individual entries have not been described in detail (though we have referenced publications describing them where available). Instead, we have focused on describing the technology behind the benchmark, the organisation of the competitions, and the rationale behind both. We have also sought to draw general conclusions about competition organisation and about the AI problem of playing platform games.

Acknowledgements

Much of this work was performed while both authors were with IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland, working under the direction of Jürgen Schmidhuber. JT was during this time supported by the Swiss Research Agency (SNF), grant number 200021-113364/1. Part of the work was supported by the Danish Research Agency (FTP), grant number 274-09-0083 (*AGameComIn*). Thanks to Jan Koutnik, Noor Shaker, Georgios Yannakakis and all of the participants in the competitions and discussions (both online in the marioai Google group and offline in conferences) for useful suggestions and feedback.

This paper incorporates some material from an earlier conference paper reporting on a first version of the benchmark and initial learning experiments [15] and another conference paper reporting on the results of the 2009 edition of the competition [27]. Compared to those papers, the current paper contains updated and technically deeper information on the benchmark, results from the 2010 competitions, descriptions of new entrants and tracks, and additional and updated discussion. More details about several of the technical aspects can be found in the first author's Master's thesis [22].

References

[1] A. Turing, "Computing machinery and intelligence," *Mind*, 1950.
[2] M. Newborn, *Kasparov Vs. Deep Blue: Computer Chess Comes of Age*. Springer, 1997.
[3] A. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal*, vol. 3, no. 3, pp. 210–229, 1959.
[4] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen, "Checkers is solved," *Science*, vol. 317, 2007.
[5] B. Sheppard, "World-championship-caliber scrabble," *Artificial Intelligence*, 2002.
[6] C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S. R. Tsai, S.-C. Hsu, and T.-P. Hang, "The computational intelligence of mogo revealed in taiwan's computer go tournaments," *IEEE Transactions on Computational Intelligence and AI in Games*, 2009.
[7] S. Lucas, "Evolving a neural network location evaluator to play ms. pac-man," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2005, pp. 203–210.
[8] S. M. Lucas, "Cellz: A simple dynamic game for testing evolutionary algorithms," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2004.
[9] G. B. Parker and M. Parker, "Evolving parameters for xpilot combat agents," in *Proceedings of The IEEE Symposium on Computational Intelligence and Games (CIG)*, 2007.
[10] J. Togelius, S. M. Lucas, H. Duc Thang, J. M. Garibaldi, T. Nakashima, C. H. Tan, I. Elhanany, S. Berant, P. Hingston, R. M. MacCallum, T. Haferlach, A. Gowrisankar, and P. Burrow, "The 2007 ieee cec simulated car racing competition," *Genetic Programming and Evolvable Machines*, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10710-008-9063-0
[11] P. Hingston, "A turing test for computer game bots," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 1, no. 3, pp. 169–186, 2009.
[12] ——, "A new design for a turing test for bots," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2010.
[13] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lönneker, L. Cardamone, D. Perez, Y. Saez, M. Preuss, and J. Quadflieg, "The 2009 simulated car racing championship," *IEEE Transactions on Computational Intelligence and AI in Games*, 2010.
[14] B. Weber, P. Mawhorter, M. Mateas, and A. Jhala, "Reactive planning idioms for multi-scale game ai," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010, pp. 115–122.
[15] J. Togelius, S. Karakovskiy, J. Koutnik, and J. Schmidhuber, "Super mario evolution," in *Proceedings of IEEE Symposium on Computational Intelligence and Games (CIG)*, 2009.
[16] K. Compton and M. Mateas, "Procedural level design for platform games," in *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE)*, 2006.
[17] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: A mixed-initiative level design tool," in *Proceedings of the 2010 International Conference on the Foundations of Digital Games*, 2010.
[18] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, "Polymorph: A model for dynamic level generation," in *Proceedings of Artificial Intelligence and Interactive Digital Entertainment*, 2010.
[19] C. Pedersen, J. Togelius, and G. Yannakakis, "Modeling player experience in super mario bros," in *Proceedings of IEEE Symposium on Computational Intelligence and Games (CIG)*, 2009.
[20] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling Player Experience for Content Creation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 54–67, 2010.
[21] N. Shaker, J. Togelius, and G. N. Yannakakis, "Towards Automatic Personalized Content Generation for Platform Games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. AAAI Press, October 2010.
[22] S. Karakovskiy, "Solving the mario ai benchmark with multidimensional recurrent neural networks," Master's thesis, University of Lugano, 2010.
[23] T. Simonite, "Race is on to evolve the ultimate mario," *New Scientist*, 2009.
[24] E. Bland, "Ai tested on "super mario" video game," *Discovery Channel News Service*, 2009.
[25] D. Leloup, "Quand c'est l'ordinateur qui joue a mario," *Le Monde*, 2009.

[26] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
[27] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 Mario AI Competition," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.
[28] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Proceedings of EvoApplications*, vol. 6024. Springer LNCS, 2010.
[29] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. in press, 2011.
[30] 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," *IEEE Transactions on Computational Intelligence and AI in Games*, 2011.
[31] J. Togelius, G. N. Yannakakis, N. Shaker, and S. Karakovskiy, "Assessing believability," in *Believable Bots*, P. Hingston, Ed. Springer, 2012.
[32] S. Bojarski and C. B. Congdon, "Realm: A rule-based evolutionary computation agent that learns to play mario," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2010, pp. 83–90.
[33] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon, "Evolving behaviour trees for the mario ai competition using grammatical evolution," in *Proceedings of EvoApps*, 2010, pp. 123–132.
[34] ——, "Reactiveness and navigation in computer games: Different needs, different approaches," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2011.
[35] E. R. Speed, "Evolving a mario agent using cuckoo search and softmax heuristics," in *Proceedings of the IEEE Consumer Electronics Society's Games Innovations Conference (ICE-GIC)*, 2010, pp. 1–7.
[36] X.-S. Yang and S. Deb, "Cuckoo seach via lvy flights," in *Proceedings of World Congress on Nature and Biologically Inspired Computing (NaBIC)*, 2009, pp. 210–214.
[37] H. Handa, "Dimensionality reduction of scene and enemy information in mario," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2011.
[38] S. Ross and J. A. Bagnell, "Efficient reductions for imitation learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.
[39] N. Shaker, S. Asteriadis, G. Yannakakis, and K. Karpouzis, "A game-based corpus for analysing the interplay between game context and player experience," in *Proceedings of International Conference on Affective Computing and Intelligent Interaction*, 2011.

Sergey Karakovskiy is a senior researcher (Ph.D. pending) at Saint-Petersburg State University. He received a 5-year diploma (2008) in Applied Mathematics and Engineering from Saint-Petersburg State University and M.Sc. degree in Informatics major in Intelligent Systems from University of Lugano in 2010. His research interests include artificial intelligence, computational intelligence in games, neuroevolution, theory of interestingness and artificial curiosity, reinforcement learning and brain-computer interface; he has published several papers on these topics in conferences and journals. He is a co-organizer of the Mario AI Championship.

Julian Togelius is an Assistant Professor at the IT University of Copenhagen (ITU). He received a BA in Philosophy from Lund University in 2002, an MSc in Evolutionary and Adaptive Systems from University of Sussex in 2003 and a PhD in Computer Science from University of Essex in 2007. Before joining the ITU in 2009 he was a postdoctoral researcher at IDSIA in Lugano.

His research interests include applications of computational intelligence in games, procedural content generation, automatic game design, evolutionary computation and reinforcement learning; he has around 60 papers in journals and conferences about these topics. He is an Associate Editor of IEEE TCIAIG and the current chair of the IEEE CIS Technical Committee on Games. He initiated and co-organised both the Simulated Car Racing Competition and the Mario AI Championship.