This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2391232, IEEE Transactions on Computational Intelligence and AI in Games

Specialization of a UCT-based General Game Playing Program to Single-Player Games

Maciej Świechowski and Jacek Mańdziuk, *Senior Member, IEEE* and Yew Soon Ong, *Senior Member, IEEE*

***Abstract*—General Game Playing (GGP) aims at designing autonomous agents capable of playing any game within a certain genre, without human intervention. GGP agents accept the rules, which are written in the logic-based Game Definition Language (GDL) and unknown to them beforehand, at runtime. The state-of-the-art players use Monte Carlo Tree Search (MCTS) together with the Upper Confidence Bounds applied to Trees (UCT) method.**

In this paper, we discuss several enhancements to GGP players geared towards more effective playing of single-player games within the MCTS/UCT framework. The main proposed improvements include introduction of a collection of light-weight policies which can be used for guiding the MCTS and a GGP-friendly way of using transposition tables. We have tested our base player and a specialized version of it for single-player games in a series of experiments using 10 single-player games of various complexity. It is clear from the results that the optimized version of the player achieves significantly better performance. Furthermore, in the same set of tests against publicly available version of CadiaPlayer, one of the strongest GGP agents, the results are also favorable to the enhanced version of our player.

Keywords*—General Game Playing, single-player games, Monte Carlo Tree Search, state-space exploration.

I. Introduction

Making machines play games has been a fascinating research topic since the dawn of Artificial Intelligence (AI), which dates back to 1950s. The goal has not been limited to creating sparring partners only, that humans could test their skills against. It has been believed that making computers play games will be a huge step towards making them actually think. The first papers published in Board Game-AI were devoted to Checkers [1] and Chess [2]. The game of Chess has even been referred to as "The Drosophila of AI" mainly due to its popularity and high intellectual appeal. Research related to other board games such as Othello [3], Backgammon [4], Scrabble [5], Pacman [6], Go [7], and more have surfaced, too. Over the years, due to exponential increase in computing speed, programs have surpassed humans in almost all popular games (the most notable exceptions being Go and Arimaa [8]), mainly due to their advantage in state-space searching capabilities. A typical scenario is that either the game is solved (e.g. checkers [9], Connect-4 [10]) or such a strong computer player is created (e.g. Deep Blue II [11] in Chess, Logistello [12] in Othello and Maven [5] in Scrabble) that research interest in a given game slowly fades.

Practically all top programs are carefully crafted to play a specific game with the help of human experts and tens of thousands of records of matches played by the most renowned players. Furthermore, much of the value of these programs stems from huge computational power of modern computers and dedicated, often hardware-based (e.g. Deep Blue II), implementation. In such an approach, there is not much space left for benefiting from the CI/AI methods, especially in terms of advancement towards the universal intelligence.

Many realized this fact and attempted to adapt more universal approaches [13], [14]. One of such trends, related to multi-game playing, commenced with Jacques Pitrat's research [15], and was subsequently continued by Michael Gherrity's SAL [16], Susan Epstein's Hoyle [17] and Barney Pell's METAGAMER [18]. The latest realization of a multi-game playing concept was proposed by Stanford Logic Group [19] under the name **General Game Playing (GGP)**. In this paper, we refer to this particular embodiment of General Game Playing. In order to encourage wider involvement from around the world, the International General Game Playing Competition has been held annually since 2005. The following players have won the title so far: Cluneplayer (2005) [20], Fluxplayer (2006) [21], CadiaPlayer (2007, 2008, 2012) [22], Ary (2009, 2010) [23], TurboTurtle (2011, 2013), and Sancho (2014). Since 2007, the winning agents use Monte Carlo simulations backed up with the so-called Upper Confidence Bounds Applied for Trees (UCT) [24], an algorithm designed to focus simulations on the most promising paths in the state-space. Our player, named MiNI-Player, was one of the contestants in the years 2012 and 2013. Like majority of the top GGP agents it relies on UCT-based simulations and similarly to many other players makes a distinction between truly multi-player games (with $N > 1$ players) and single-player ones (puzzles) with $N = 1$. Separate instances of the MiNI-Player differing in their underlying concepts is applied to each of the two cases. The first case is addressed in [25] along with a detailed description of MiNI-Player implementation. This paper is devoted to the case of single-player games. The paper is organized as follows. Section II extends the introduction by providing background of General Game Playing. Section III points out the reasons why single-player games should be handled separately from

Maciej Świechowski is a PhD Student at Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland. e-mail:m.swiechowski@ibspan.waw.pl

Jacek Mańdziuk is a professor at the Faculty of Mathematics and Information Science, Warsaw University of Technology, Koszykowa 75, 00-662 Warsaw, Poland. e-mail:mandziuk@mini.pw.edu.pl

Yew Soon Ong is a professor at the School of Computer Engineering, Nanyang Technological University, Block N4, 2a-28, Nanyang Avenue, Singapore 639798 e-mail:ASYSOng@nt.edu.sg

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2391232, IEEE Transactions on Computational Intelligence and AI in Games

multi-player ones. In section IV a brief reference to the MiNI-Player's baseline routine, which is adapted to the case of puzzles, is presented. Sections V and VI cover the changes introduced by implementation of this adaptation. The next two sections are the results and comments on the related work, respectively. Section IX concludes the paper.

II. General Game Playing

Games in GGP are described in a formal language called Game Description Language (GDL) [26], which defines their rules. In order to understand the rules at runtime, players must use an inference engine. The GDL is a declarative, predicate logic language based on Datalog, which is syntactically a subset of Prolog. GDL allows to describe any game from the genre of *multiplayer*, *finite* (number of players, states and actions), *deterministic*, *synchronous* games. Although in *synchronous* games, players perform moves simultaneously and the next state in a game is a result of the combined moves from all participants, it is possible to simulate turn-based games in GDL due to the possibility of no-operation moves. Compared to the plain Datalog, GDL introduces keywords, a predefined set of clauses consisting of: *role*, *init*, *legal*, *does*, *next*, *true*, *goal*, *terminal* and *distinct*. For the detailed GDL description, please consult its specification [26] or an on-line tutorial [27]. As an example we present below an excerpt, containing only the major rules, from the GDL-based description of a puzzle Hanoi.

```
(role player)
(init (on disc5 pillar1))
(init (on disc4 disc5))
(...)
(init (on disc1 disc2))
(init (clear disc1))
(init (clear pillar2))
(init (clear pillar3))
(init (step s0))

(<= (legal player (puton ?x ?y))
    (true (clear ?x))
    (true (clear ?y))
    (smallerdisc ?x ?y))
(<= (next (step ?y))
    (true (step ?x))
    (successor ?x ?y))
(<= (next (on ?x ?y))
    (does player (puton ?x ?y)))
(<= (next (on ?x ?y))
    (true (on ?x ?y))
    (not (put_on_any ?x)))
(<= (next (clear ?y))
    (true (on ?x ?y))
     (put_on_any ?x))
(<= (next (clear ?y))
    (true (clear ?y))
    (not (put_any_on ?y)))

(...)

(<= (goal player 100)
    (tower pillar3 s5))
(<= (goal player 80)
    (tower pillar3 s4))
(...) (<= (goal player 0)
    (tower pillar3 ?height)
    (smaller ?height s2))

(...)

(<= terminal
    (true (step s31)))
(<= (tower ?x s0)
    (true (clear ?x)))
(<= (tower ?x ?height)
    (true (on ?y ?x))
    (disc_or_pillar ?y)
    (tower ?y ?height1)
    (successor ?height1 ?height))
```

In GGP, the game rules are communicated to the players by the Gamemaster (GM) as part of a *START* message. The GM serves roles of an arbiter and communication hub in GGP. It collects moves from the players and sends back an information about all chosen actions. If a participant submits an illegal move, the GM randomly chooses a legal one instead. There are two time constraints in GGP: a *START CLOCK* and a *PLAY CLOCK*. The first one defines time interval between the start of a game and the first required action (an initial thinking time), whereas the second one represents the time allotted for each action (move generation). The clocks are related to the messages *START* and *PLAY*, respectively. There is also a *STOP* message, which is sent by a GM in a terminal state (which as the name implies ends the game).

III. Puzzles

The reader may wonder why we propose a different approach for puzzles from all other types of games. We justify this decision and bring its motivation in this section. Firstly, compared to other types of games, there are no opponents in puzzles, hence there is no need for performing the opponents' modeling (which is generally a crucial issue in GGP). Secondly, puzzles are usually very hard to play well. For example, in two-player games it is only necessary to play better than the opponent in order to win. Sometimes a robust search with a simple playing policy is sufficient, especially when the opponent plays poorly or at the same level. In one-player games, however, obtaining a maximal score means solving the puzzle. The course of actions leading to a victory is usually extremely difficult to find for non-trivial games. If we play randomly, we are almost guaranteed to obtain the lowest possible score. In other words, a full game tree usually contains a few or only one path leading to the maximal score. Therefore, a basic Monte Carlo Tree Search is not really well-suited for such games and using some kind of heuristic-driven search is inevitable. Thirdly, although it is difficult to find an optimal solution, once a partial solution is found during the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2391232, IEEE Transactions on Computational Intelligence and AI in Games

state-space search process, it is easy to reproduce. There are no external factors such as the opponents or non-determinism which could prevent us from reaching the state again once visited. In particular, if we find a full solution, i.e. with a maximal score, we can stop searching and just remember the list of actions which lead to that solution.

Finally, please note that the number of players involved in a game is easy to detect by counting the number of **role** facts present in the game description.

IV. Base-line Approach

Before focusing on the proposed modification tailored for single-player games let us summarize the base-line approach. Please, refer to [25] for the details.

A. Monte Carlo Tree Search

Monte Carlo algorithms have been successfully applied in various domains such as statistical physics, computational mathematics, numerical integration, finances or business. The Monte Carlo Tree Search (MCTS) algorithm consists in searching the game tree by means of performing random simulations. The method proved to be successful in games for which effective heuristics are not known (such as Go [7]) as well as in imperfect information games (such as Magic The Gathering [28]). MCTS is also the basis of the state-of-the-art GGP solutions since 2007, which confirms its efficiency and pertinence. There are four traits which contribute to the overall high performance of MCTS in GGP. Namely, the method is:

1) **Knowledge-free.** No game-specific knowledge is needed so the method is well-suited for generally defined applications such as GGP. The only requirement of the MCTS method is the knowledge of how to simulate a game (traverse the game tree) while assuring asymptotic convergence to an optimal play. This feature is of particular importance when considering the fact that a design of an algorithm capable of deriving autonomously a suitable evaluation function in GGP (as well as in any single, non-trivial game) without human intervention is an open problem, still far from being satisfactorily addressed.
2) **Anytime.** The method can be stopped at any time and return the currently most promising continuation to play. It is therefore appealing for scenarios with time constraints imposed on action-taking. In addition, there is no need to impose a limit on the search depth, which is a non-trivial task in GGP as games have varying branching factors and PLAY CLOCKs. In its underlying concept the MCTS resembles a variant of an iterative best-first search method.
3) **Asymmetric.** The game tree is built in asymmetric way such that rewarding actions are chosen more frequently.
4) **Scalable.** Since simulations are performed in an independent manner, the method can be efficiently parallelized on multi-core systems or clusters.

The MCTS, as depicted in Figure 1, consists of four phases: **selection**, **expansion**, **simulation** and **back propagation**.

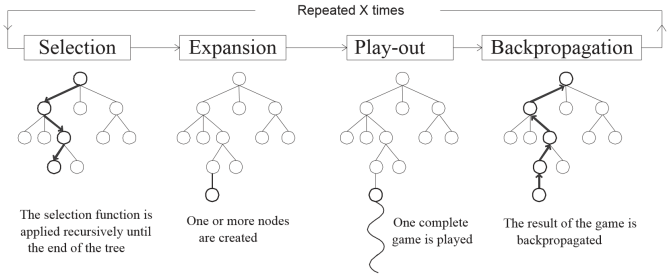

Fig. 1. Monte Carlo Tree Search phases. The figure is reproduced from [29]

The goal of the method is to iteratively build a game tree by adding new nodes and store the simulated average game result (score) in each node. In the **selection** phase, an already-built part of a tree is recurrently traversed starting from the root until a leaf node is reached. At each level the most promising child is selected. In the **expansion** phase, provided that a state associated with the last visited (leaf) node is not terminal, N new child nodes (typically $N = 1$) are allocated and states associated to them are added to the tree. In the **simulation** phase, as many as time permits, complete game-simulations from the newly-added state(s) to terminal (end of game) states are performed by means of a random move selection on behalf of all the players. Each game result obtained in such a simulation is **propagated back** to all nodes visited on the playing path, up to the root node. In each node the cumulative result, the average result and the number of visits to that node are stored.

B. The UCT algorithm

The Upper Confidence Bounds Applied To Trees (UCT) method [24] is a policy used in the selection phase in order to select the most promising child node. It is a popular tree-search algorithm applicable also beyond game domain. The UCT maintains the ratio between the exploitation of higher-rewarded nodes and the exploration of less visited ones. In this method an action a^* is selected according to the following formula:

$$a^* = arg \max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln\,[N(s)]}{N(s,a)}} \right\} \qquad (1)$$

where *a* - is an action; *s* - is the current state; *A(s)* - is a set of actions available in state *s*; *Q(s,a)* - is an assessment value of performing action *a* in state *s*; *N(s)* - is a number of previous visits of state *s*; *N(s,a)* - is a number of times an action *a* has been sampled in state *s*; *C* - is a coefficient defining a degree to which the second component (exploration) is considered. The action-selection scheme asymptotically converges to a min-max/alpha-beta search [30].

C. Simulation Strategies

The main novelty of our method, compared to classical UCT-MCTS implementation, lies in the introduction of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2391232, IEEE Transactions on Computational Intelligence and AI in Games

simulation strategies which are aimed at optimization of the simulation (MCTS) phase. A strategy is assigned to a player in each simulation and provides a rule according to which their actions should be chosen. However, we would like to avoid deterministic behavior since it would undermine the idea of MCTS. For this reason, at each move (in each node), there is a probability P of making a move according to the assigned strategy, or randomly, otherwise. Parameter P is tuned individually for each strategy and ranges from 0.7 to 0.95. Please refer to [25] for an exhaustive specification of the strategies. Their brief description is presented below:

1) **Random.** Self-explanatory - actions have equal probability of being chosen. It is the simplest and the fastest strategy. Effectively, when combined with the MCTS simulations, it leads to a random search.
2) **Approximate Goal Evaluation (AGE).** We approximate the degree of truth of the *goal* relation. The *goal* rules are filtered to leave only those which can produce (make true) a fact with the highest obtainable score to our role. The idea is to greedily take actions which lead to states closer to this goal. We were inspired by the method used in the Fluxplayer's evaluation function [21], but we do not use fuzzy logic norms (t-norms and s-norms) as Fluxplayer does. In order to compute a degree of fulfilment of the goal condition we introduce a concept of the so-called OR-AND proof trees, which are part of our GDL inference engine [31]. The procedure of computing the AGE value is recurrent - it starts from the root but the actual values are back-propagated from the leaves. Basically for each fact we assign 1 or 0 depending on whether it holds in the current state. For each rule (AND node) we count how many conditions are true divided by how many conditions are defined for the rule. When gathering facts from all sources (OR nodes) we use maximum over all values gathered from the AND nodes. When the values are equal a simple tiebreaking procedure is applied (see [25] or [31] for the details).
3) **Statistical Symbols Counting (SSC).** Here, we will use the term relation as a container for facts which share a common name - they can be produced by rules or explicitly defined. For each relation, this strategy monitors the average number of facts which are true in a given state. Also, for a triplet (relation; index of an argument in the relation; symbol from the domain of the argument) the strategy monitors the number of occurrences of the symbol. For example, if $(cell11red)$ denotes a fact in a game, the relation would count the number of 1 symbols appearing as the first and the second argument and red symbols as the third one. The same goes for every other symbol in a domain. Each computed value is tested for a correlation with the results of the game. If such a correlation exists, the respective symbol is assigned a specifically calculated weight and becomes part of the evaluation function which can be used since the first PLAY message. SSC is the only strategy which involves a learning-phase thus not being enabled from the very beginning (right after the START message). For more information please consult [32] or [25].
4) **Mobility.** The aim is to choose moves which maximize the relative number of available actions compared to the opponents' actions counts. This strategy reflexes the usually valid assumption that the more actions are available to the player the better.
5) **Exploration.** The aim is to select actions leading to the states which are different from those visited before.
6) **History Heuristic.** This is a popular heuristic in game domain. It assumes that actions which proved to be generally promising (strong) - regardless of the states in which were played - are chosen more frequently.
7) **Score.** Used only if the GDL description. It follows a typical pattern of defining scores for players and having the higher score is a condition of *goal* rules. Actions which greedily maximize the score for our player have greater priority. This is a new strategy, not included in [25].

If there are several actions tied for being best from the point of view of a given strategy, a random one out of them is chosen. A strategy is responsible for choosing actions during a simulation but there is also a decision to be made at higher level, i.e. which strategy should be assigned to a given simulation. We tested a few mechanisms of such a selection and found the Upper Confidence Bounds (UCB) formula [33] to be the most effective. UCB is a flat variant of UCT, in which the whole decision space can be restricted to only one level in a tree. Chronologically, it was the UCT algorithm which was proposed as an extension to UCB.

D. Modified formula for move selection

Among distinct features of MiNI-Player there are custom inference engine, specific parallelization scheme and modified move selection formula. The first two traits remain unchanged for single-player games so they are of lesser importance to the scope of the paper. Move selection formula is applied in the actual game (during every *PLAY-CLOCK* time) and should not be confused with actions performed during internal simulations. A straightforward approach is to select action leading to a state with the best average score ($Q(s,a)$ in (1)). Our action selection scheme is a bit more complex, based on - what we call - an Effective Quality (EQ). In a given decision context (node), let us define by *Our* - the number of moves available to our role and by *Total* - the number of all joint (tuple) moves available. In order to calculate EQ value we distinguish four types of nodes among the next move candidates (see Table I). For each child *node* of the root node (i.e. each candidate move) we calculate its EQ according to Table I. A *node* with the highest EQ value is chosen. If more than one *node* is tied with this value we choose an action having the highest History Heuristics (HH) value. Since all strategies record their history, HH provides an additional mechanism for the final move recommendation. If some nodes are still tied, one of them is selected at random. When applying the above procedure our agent analyzes not only the child

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2391232, IEEE Transactions on Computational Intelligence and AI in Games

TABLE I. EQ VALUE BASED ON THE ROOT CHILDREN *node* PROPERTIES.

Condition in a *node*	**EQ formula**
node.Terminal = True	$EQ = node.Q$
Our = 1	$EQ = \min_{i=1\ldots N}(node.Child[i].Q)$
1 < Our < Total	$EQ = \frac{\min_{i=1\ldots N}(node.Child[i].Q)+node.Q}{2}$
Our = Total	$EQ = \max_{i=1\ldots N}(node.Child[i].Q)$

nodes of the root but also their child nodes (grandchildren of the root), so, effectively, the tree is searched one level deeper.

V. SIMULATION STRATEGY FOR PUZZLES

Adaptation of strategies on the basis of their average simulation scores works very well for majority of multi-player games. Therefore, at our first attempt, we applied the multi-player games approach to single-player ones without introducing any modifications. This solution, however, turned out to be relatively weak. The reason is that for most, if not all, non-trivial puzzles available in GDL repositories [34], [35], [36], a common pattern of scores' distribution can be observed. The first portion of simulations end with zero scores, since no (partial) solution is found. This early phase can be measured in several thousands of Monte Carlo roll-outs or can even last through the whole game in the case of the most complex puzzles, i.e. no positive scores are ever achieved. Once the first non-zero result of a simulation is added to the UCT tree, the subsequent simulations are either able to repeat it by applying the same line of play or continue to miss. However, if using some strategies leads to the same score, these strategies are indistinguishable by the strategy-selection rule and all them are used evenly. As a result, since simulations are evenly split among strategies, it is very likely that many of them are wasted by using ineffective strategies.

Given enough time, presumably after thousands of simulations, a new higher score is likely to be found. This is a kind of pattern in which the scores start with the minimal value and occasionally shift to a better one until the highest possible score is found (which is not guaranteed to happen in a reasonable amount of time). There is no variety in such distributions which is probably caused by the lack of tactical plays which are often part of multi-player games. In puzzles, everything is down to finding the next (better partial or complete) solution. Therefore, simulations should be geared towards searching the states that are highly distinct from the previously visited ones, so as to increase the chance of finding a winning state. The Random strategy becomes a natural candidate here, since it enables diverse and fast simulations leading to various states being visited. We re-evaluated all the strategies, used so-far in multi-player games, in the context of puzzles and came to the following conclusions:

1) **HH and SSC.** These strategies rely on average scores and, therefore are useless until any partial solution is found. Furthermore, once such a solution is found, there is no need to rely on its average score, because it can easily be repeated at any time. Both strategies are clearly not suitable for one-player games.
2) **AGE and Score.** These two strategies were discarded as well. It seems that puzzles are to complex for those rules to work. Both strategies are defined using a precise rule which leads to "quite deterministic" playing where many actions are frequently repeated.
3) **Mobility.** In puzzles, this is simplified to increasing the number of available moves. In preliminary experiments this approach appeared to be ineffective.
4) **Exploration.** This is the winning strategy - very well suited for single-player games.
5) **Random.** A portion of purely random simulations is desirable to be mixed with some kind of heuristic. Random strategy fulfills the requirement of variety, offering high performance speed as an added incentive.

Ultimately, the MiNI-Player version tuned for single-player games uses the **Random** strategy and **six variants of Exploration** strategy. These variants are discussed in the remainder of this section.

Let us first introduce a basic notion of a difference between any two game states S_1 and S_2:

$$diff(S_2, S_1) = |(S_2 \setminus S_1)| + \delta|(S_1 \setminus S_2)|, \delta \in \{0, 1\} \quad (2)$$

A difference in (2) can be either symmetrical or non-symmetrical depending on the value of delta. In a symmetrical case, we sum up the number of state facts (predicates) which belong to S_1 and not to S_2 and the number of facts which are in S_2 and not in S_1. We would rather not use any values for delta other than 0 and 1 since such choices would be lacking natural intuitive interpretation. Any value between 0 and 1 would mean weighting the two differences (symmetric and asymmetric). Instead of weighting, we prefer to keep them separately, since one or the other may prove useful for a given game depending on how the state logic is defined in a particular GDL description.

It is worth recalling that a game state consists of GDL atomic facts such as $[cell\ 1\ 1\ b]$ or $[control\ xplayer]$. Only dynamic facts, i.e. those which are affected by *init* and *next* keywords, count as the building-blocks of a state.

In the current state S_k, the algorithm, for each legal action a_i, computes the next state $S^i_{k+1} = next(S_k, a_i)$ which is the state obtained by applying action a_i. Now, for each S^i_{k+1} the most similar one among the previous N states visited in the simulation is found with the following formula:

$$minDiff(S^i_{k+1}, N) = min^k_{j=k-N+1} diff(S_{k+1}, S_j) \quad (3)$$

Finally, we choose an action which maximizes the minimal difference measure over all possible next-states:

$$a = argmax_i(minDiff(S^i_{k+1}, N)) \quad (4)$$

The minimal difference measure ($minDiff(S^i_{k+1}, N)$) introduced in equation (3) can be regarded as a difference between a state and set of states. We do not aim at maximizing the difference between a new state and the current state only, i.e. applying a greedy approach. A motivation behind the Exploration strategy is to search for states which are generally

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2391232, IEEE Transactions on Computational Intelligence and AI in Games

different from the recently visited states. A margin of only one state, which can be obtained by assigning $N = 1$, is too narrow and causes problems with falling into local minima. For example, if two very different states are reachable one from another, setting $N = 1$ would result in periodical visiting of these two states.

Six variants of the Exploration Strategy come from various settings of the parameters: $N \in \{4, 6, 8\}$ and $\delta \in \{0, 1\}$. Together with a Random one, each strategy receives a seventh simulation to perform.

VI. Tree Construction Scheme for Puzzles

In this section the key differences between multi-player and single-player games in terms of building and expanding the UCT tree are presented.

A. Selection Phase

One of the major changes related to one-player games is modification of the UCT formula (1). In its original formulation, UCT stores the average score of each possible action in a node. This average is computed by dividing a cumulative result propagated through this node by a number of times the node was visited. This component represents an expected outcome of applying action a in state s in (1). In single-player games scenario, the average score is of no use since repeating any simulation path previously traversed is a trivial task. We, therefore, changed the formula so that it uses a maximum score - $MAX(s, a)$, instead:

$$a^* = arg \max_{a \in A(s)} \left\{ MAX(s,a) + C\sqrt{\frac{ln\,[N(s)]}{N(s,a)}} \right\} \quad (5)$$

B. Expansion Phase

The second change (considered by several other top players, as well consists in keeping a local memory of performed actions in each simulation, in case a solution is found. We found storing actions' indices only a sufficiently efficient solution. There is also a global maximum score $MAX \in [0, 100]$ that is being kept track upon. Every time a simulation finishes with a better result than the previous value of MAX, the UCT tree is expanded by adding the whole branch of states visited during that simulation until a terminal state is reached. If a complete solution, i.e. the one with the maximum score is found, no more simulations are performed. This approach works quite well with the so-called root parallelization [37]. The best solution so-far is inserted into a tree, so it can be easily reproduced, action by action, without making any adjustments to the root parallelization scheme.

C. State Transpositions

The third considered modification is the use of transposition tables (TT) [38]. TT is a well-established optimization technique aimed at speeding up the search of the game tree and reducing memory overhead by grouping states reachable from more than one position. This approach performs particularly well in single-game programs where a dedicated efficient state representation is available. In GGP, however, TT impose an overhead which in some games may lead to a slower overall search than when they are not used. While in multi-player games the answer whether it is worth using TT depends mainly on a particular game being played, in puzzles it is almost always beneficial to use this optimization. Most of the single-player games are defined in such a way that there are many ways of reaching a particular state ((including the case of a state cycle of length 2, i.e. reaching two states A and B alternately a number of times)), which makes such games good candidates for using TT. There have been a few efficient techniques for maintaining state transpositions proposed in games literature. The most popular one is the Zobrist hashing (ZH) [39], which is applied to chess-like board games. However, there are several reasons why the ZH cannot be applied in a general case of GGP games. Firstly, the method assumes that the current state (the board) is defined by a single structure. In GGP, however, there can be none, one or multiple boards or any number of other/various structures. Secondly, an efficient implementation of the ZH requires dimensions of the structure to be known beforehand.

Therefore, instead of using ZH, we propose a multi-resolution hash table to resolve state transpositions. In state transpositions a typical task is to find a transposition for a given state or return an answer that no transposition for the given state has been visited yet. Instead of using a single-key hashmap we use a multi-resolution hash table consisting of five levels. Whenever we present a new state in the input, five properties of that state are quickly computed. Each unique combination of the values of these properties has its own single-key hashmap representation. At the highest level, states are grouped together into buckets based on their properties:

- Total number of legal actions in a state
- Total number of state facts a state is composed of
- A number of non-empty types of facts (a type such as "cell" or "step" is non-empty if it contains at least one fact in the current state, e.g. (cell a 2 white)).
- A number of facts in each relation
- A vector of hash codes in each relation

The algorithm can be regarded as a decision tree. At each level, in a non-leaf node, a decision which node to choose based on the value of the property is made. At the leaf level, the hashmaps with the actual keys are placed.

Hashing is implemented via C# *Equals/GetHashCode* interface. Each fact is a vector of integers, since each symbol is converted to a 16-bit integer number in our system. The hashing function is presented in equation (6):

$$hash = \sum_{r \in R} \sum_{i=1}^{N} (2^i * I(r_i)) \quad (6)$$

where: R - denotes the set of groundings for a given relation; $r \in R$ - denotes a single grounding; N is the relation's arity; r_i - is a symbol which appears at the i-th position (index) of relation r; and $I()$ is a coding function which converts symbols to 16-bit integers.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2391232, IEEE Transactions on Computational Intelligence and AI in Games

As long as two states differ in at least one out of these five properties, it is guaranteed that there will be no collision even if they get the same value from the hashing function. This is due to the fact that such states belong to different hashmaps. We have found that collisions are very rare in our multiresolution approach at the cost of a slight computational overhead. However, checking state transpositions is performed only in the UCT part of the MCTS algorithm which takes significantly less time than the Monte Carlo part. The latter is still the bottle neck.

In the back propagation phase, we check for transposed states in each visited node and if there are any, we also update the score and visits count in them.

VII. Empirical Results

In this section we present a summary of results divided into three subsections. The first one of them is devoted to comparison of the proposed approach with a baseline MCTS/UCT player. The next subsection presents a discussion on the issue of efficient calibration of the internal parameters of the proposed search strategies. The main experimental results are presented and discussed in Subsection VII-C, where we compare our proposed approach with the baseline approach (i.e. our player without the enhancements for single-player games) and with CadiaPlayer - a three-time GGP Competition winner. In this comparison the full matches are played, with both START and PLAY clocks, in the spirit of the GGP Competition protocol.

A. Long-time comparison and selection of strategies

Here, in each instance of the experiment, we run the tested players for long time settings, up to 10 minutes, with the START clock phase only. At certain timestamps, i.e. the first second and every minute, we observed the highest score obtained so far and the simulation strategy, if applicable, which had found that score. The instance (single experiment) was terminated if the maximum possible score was found or the START clock expired. Each player in the experiment played 100 instances of each game. For the sake of clarity of the presentation and due to the lack of space the results are presented for the selected timestamps only. For the same reasons the exact confidence intervals are not provided in the tables (on a general note, all of them were not greater than ± 6).

In Table II, we compare four players over the 10-minute timespan. The first two players are MAX and AVG - both implementing the proposed solution, with 6 variants of Exploration strategy, differentiated only by application of the maximum or average operators, respectively to the assessment of $Q(s, a)$ in the UCT formula. The ALL player uses the above-mentioned 6 variants of Exploration strategy in addition to all the remaining strategies introduced in Section IV-C. The last player, R, is a plain MCTS/UCT approach with random simulation strategy used in the Monte-Carlo phase. All players except for the AVG apply the maximum operator in the UCT formula. Since the MAX player is our proposed enhancement, we are particularly interested in direct comparisons between MAX and the other players. The results support the following conclusions.

- **MAX is slightly better than AVG.** Generally speaking, the results of MAX and AVG are very close. The only significant advantages of MAX are observed in Hamilton (for 1m time), Hanoi (3m) and Rubik's Cube (10m). In the remaining (game, time) pairs, these two approaches are statistically indistinguishable.
- **MAX is better than R.** The random player performs equally good to MAX in Buttons only, which turned out to be the simplest game. In all other games, there exist time frames where MAX is significantly stronger than the random player and there are no time frames where R is stronger than MAX. This property holds true in the whole 10-minute interval.
- **MAX is stronger than ALL.** The MAX player outperforms the player which uses all simulation strategies in Circle Solitaire (1s), Eight Puzzle (1m), Hamilton (1m), Hanoi (1-3m), Lights Out (3m), Rubik's Cube (3m+) and Untwisty Corridor (1-3m). The players are comparably good in Buttons, Hunter and Knights Tour. It is worth noting that for the longest time (10m) the difference becomes statistically indistinguishable in all games but Rubik's Cube. This observation suggests that the ALL player is simply converging slower due to the computational burden imposed by using all strategies.

In addition, in order to justify the choice of the Exploration strategy as the main one, we measured how many times a particular strategy was the highest score founder at the respective time frame. These measurements, presented in Table III, were performed using the ALL player. Since there are six variants of Exploration, we present both the accumulated counts and the average counts per each variant.

The results show that the Exploration strategy finds the highest score the most frequently at higher time frames. At lower time frames, however, the Approximate Goal Evaluation (AGE) strategy is the winner. In such a complex scenario as MCTS it is not clear whether frequent finding of the highest score means that the simulation policy which found the score is best-suited for a game. Since all strategies contribute to the development and search of the same game tree, strong synergy effects are highly possible (e.g. due to previous simulation passes, a currently chosen strategy may have an advantage of starting from a privileged state, located close to the goal state). Nevertheless, the significant usage of the AGE in the top score founders suggests that this strategy could have been prematurely discarded by us. We will investigate incorporation of AGE into the final solution (our competitive MiNI-Player version) in the future.

B. Calibration of the Exploration Strategies

One of the critical parameters of the proposed method is the selection of depths in the Exploration strategy. In the preliminary simulations we found out that 8 is the greatest feasible depth in terms of computational efficacy. It also turned out that using two consecutive exploration depths results in quite similar playouts and that the effects of using low values

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2391232, IEEE Transactions on Computational Intelligence and AI in Games

TABLE II. A COMPARISON OF THE HIGHEST SCORES ACHIEVED BY FOUR PLAYERS AT CHOSEN TIME FRAMES. THE **MAX** AND **AVG** PLAYERS ARE THE PROPOSED SOLUTION DIFFERENTIATED ONLY BY THE OPERATOR (max AND $average$, RESPECTIVELY) WHICH IS APPLIED TO THE ASSESSMENT OF $Q(s,a)$ IN THE UCT FORMULA. THE **ALL** PLAYER, IN ADDITION TO THE PROPOSED 6 VARIANTS OF EXPLORATION, USES ALL THE REMAINING SIMULATION STRATEGIES. **R** IS A PLAIN MCTS/UCT PLAYER USING ONLY RANDOM SIMULATIONS IN THE MCTS PHASE. THE SCORES WERE CAPTURED AT SELECTED TIME FRAMES AND AVERAGED AFTER 100 REPETITIONS.

Time:	**1s**				**1m**				**3m**				**10m**			
Game \ Player:	**MAX**	**AVG**	**ALL**	**R**	**MAX**	**AVG**	**ALL**	**R**	**MAX**	**AVG**	**ALL**	**R**	**MAX**	**AVG**	**ALL**	**R**
8-Puzzle	0	0	0	0	99	99	0	0	99.03	99.08	99	0	99.03	99.08	99.07	58.9
Buttons	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Circle Solitaire	99	100	48	0	100	100	100	100	100	100	100	100	100	100	100	100
Hamilton	0	0	0	0	98.8	91.67	90	56.9	100	100	97.49	90	100	100	100	97
Hanoi	0	0	0	0	60	59.8	34.4	36.4	79.8	60	55.6	59.6	80.4	78	80	70.4
Hunter	0	0	0	0	87	87	87	75	87	87	87	87	87	87	87	87
Knights Tour	0	0	0	0	90	90	90	70.8	98.84	95.4	95.58	71.13	99.44	99.01	99.43	72.63
Lights Out	0	0	0	0	0	0	0	0	100	100	0	0	100	100	100	22
Rubik's Cube	0	0	0	0	0	0	0	0	17.85	18.85	3.2	0	42.2	37.05	27.9	9.95
Untwisty Corridor	0	0	0	0	10	9	0	0	96	97	90	62	100	100	100	99

TABLE III. A BREAKDOWN OF STRATEGIES BY THE NUMBER OF TIMES THEY FOUND THE HIGHEST SCORE IN PARTICULAR TIME FRAMES. THE DATA WAS CAPTURED DURING PLAYOUTS OF THE (ALL) PLAYER, WHICH USES ALL DISCUSSED STRATEGIES INCLUDING THE 6 VARIANTS OF EXPLORATION. FOR EACH PAIR (GAME, TIME) THE EXPERIMENT WAS RUN 100 TIMES.

Game	**1s**	**1m**	**3m**	**5m**	**10m**
No score found	852	400	169	101	0
Random	3	27	30	25	25
AGE	46	84	94	93	106
History Heuristic	4	43	68	70	74
Mobility	12	47	53	59	55
SSC	0	0	13	21	18
Score	5	9	21	26	33
Exploration Total	78	390	552	605	689
Exploration Average	13	65	92	100.8	114.8

$1, 2$ are similar to those of random simulations. Therefore, in the final simulations we were inclined to use depths between 3 and 8 evenly scattered in this range. With these restrictions in mind, we compared three versions of the Exploration-based player. The first variant (EXP468) employed depths of 4,6 and 8, the second one (EXP357) employed depths of 3, 5 and 7, whereas the last one (EXP1-8) employed all depths from 1 to 8.

The results are presented in Table IV. Using all depths from 1 to 8 proved to be suboptimal to the other variants, losing in Circle Solitaire (1s), 8-Puzzle (1m), Hamilton (1m), Hanoi (3m), Lights Out (3m), Rubik's Cube (10m) and Untwisty Corridor (1m). The EXP468 and EXP357 players achieved equally good results within the statistical error margin, so both could be applied. Our final (arbitrary) decision was to use depths $4, 6$ and 8 in the final experimental setup.

C. Final Results

In order to verify the efficacy of proposed solution and to justify the choice of algorithms presented in this paper a comparison among the three following players was made. The first player, called **Dedicated MINI-Player** (DMP), is the version of MiNI-Player introduced in this paper, i.e. the one equipped with single-player games enhancements. The second one, named **Baseline MINI-Player** (BMP) is a regular MiNI-Player, i.e. a version dedicated to multi-player games. This variant of MINI-Player does not use the enhancements introduced in sections V and VI. The third contestant, is a version of the **CadiaPlayer** (CP) - a three-times GGP Competition winner - publicly available for download [40]. Players have been tested in a series of 10 games of various complexity and three choices of (START, PLAY) clock pairs, determining the time allotted for an initial preparation and for making a move, respectively. The motivation was to provide some insight into how fast the simulations performed by the players may converge. Moreover, in a competition scenario, the actual time for move is not known in advance. Games definitions are available in GGP Internet repositories [35], [36], [34].

The results, averaged over 100 trials in each game, are shown in tables V, VI and VII for the $(120, 30)$, $(45, 15)$ and $(2, 1)$ (START, PLAY) clock pairs, respectively.

TABLE V. A COMPARISON OF THE PLAYERS UNDER THE LONGEST CLOCK TIMES. THE START-CLOCK IS SET TO 120 SECONDS AND THE PLAY-CLOCK IS EQUAL TO 30 SECONDS. EACH PLAYER WAS TESTED ON 100 INSTANCES OF EACH GAME.

Game	**Dedicated MINI-Player**	**Baseline MINI-Player**	**CadiaPlayer**
8-Puzzle	99.76	98.01	1.98
Buttons	100.00	100.00	100.00
Circle Solitaire	100.00	100.0	100.00
Hamilton	100.00	91.10	100.00
Hanoi	82.40	80.00	100.00
Hunter	87.00	87.00	87.00
Knights Tour	100.00	100.00	100.00
Lights Out	99.00	9.00	0.00
Rubik's Cube	10.50	0.00	35.00
Untwisty Corridor	100.00	100.00	100.00
Average	**87.86**	**76.51**	**72.40**

8-Puzzle and **Lights Out** are complex enough games that none of the players can find a positive score with the shortest clocks' settings. In longer-lasting experiments, DMP is superior to other players being the only one capable of achieving the highest scores regularly. Please note that 8-Puzzle have three scores defined: 0 - no solution was found; 99 - a solution was found between steps 31 and 60; 100 - a solution found in the optimal number of 30 steps. In this respect, the difference between 99.00 and 99.74 in the average results

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2391232, IEEE Transactions on Computational Intelligence and AI in Games

TABLE IV. A COMPARISON OF THE HIGHEST SCORES ACHIEVED RESPECTIVELY BY THE PLAYER USING EXPLORATION STRATEGY WITH DEPTHS 4,6 AND 8 (**EXP468**) - THE PROPOSED SOLUTION, THE PLAYER WITH THE EXPLORATION DEPTHS EQUAL TO 3,5 AND 7 (**EXP357**) AND THE ONE WHICH USES ALL DEPTHS FROM 1 TO 8 INCLUSIVE (**EXP1-8**). THE SCORES WERE CAPTURED AT SELECTED TIME FRAMES AND AVERAGED OVER 100 TRIALS.

Time:	**1s**			**1m**			**3m**			**10m**		
Game \ Player:	**EXP468**	**EXP357**	**EXP1-8**	**EXP468**	**EXP357**	**EXP1-8**	**EXP468**	**EXP357**	**EXP1-8**	**EXP468**	**EXP357**	**EXP1-8**
8-Puzzle	0	0	0	99	89	0	99.03	99.07	99	99.03	99.07	99.03
Buttons	100	100	100	100	100	100	100	100	100	100	100	100
Circle Solitaire	99	98.57	81	100	100	100	100	100	100	100	100	100
Hamilton	0	0	0	98.8	98.9	91.3	100	100	100	100	100	100
Hanoi	0	0	0	60	60	59.8	79.8	80	60	80.4	80.2	80
Hunter	0	0	0	87	87	87	87	87	87	87	87	87
Knights Tour	0	0	0	90	90	90	98.84	96	96	99.44	99.6	99.4
Lights Out	0	0	0	0	0	0	100	99	0	100	99	100
Rubik's Cube	0	0	0	0	0	0	17.85	18	15.05	42.2	41.65	33.45
Untwisty Corridor	0	0	0	10	10	0	96	98	93	100	100	100

TABLE VI. A COMPARISON OF THE PLAYERS UNDER THE MIDDLE-LENGTH CLOCK TIMES. THE START-CLOCK IS SET TO 45 SECONDS AND THE PLAY-CLOCK IS EQUAL TO 15 SECONDS. EACH PLAYER WAS TESTED ON 100 INSTANCES OF EACH GAME.

Game	**Dedicated MINI-Player**	**Baseline MINI-Player**	**CadiaPlayer**
8-Puzzle	99.01	13.86	0.00
Buttons	100.00	100.00	100.00
Circle Solitaire	100.00	100.00	100.00
Hamilton	100.00	90.30	99.90
Hanoi	80.20	80.00	97.40
Hunter	87.00	87.00	87.00
Knights Tour	100.00	86.50	100.00
Lights Out	35.00	4.00	0.00
Rubik's Cube	0.00	0.00	35.00
Untwisty Corridor	97.00	92.00	100.00
Average	**79.82**	**65.37**	**71.93**

TABLE VII. A COMPARISON OF THE PLAYERS UNDER THE SHORTEST CLOCK TIMES. THE START-CLOCK IS SET TO 2 SECONDS AND THE PLAY-CLOCK IS EQUAL TO 1 SECOND. EACH PLAYER WAS TESTED ON 100 INSTANCES OF EACH GAME.

Game	**Dedicated MINI-Player**	**Baseline MINI-Player**	**CadiaPlayer**
8-Puzzle	0.00	0.00	0.00
Buttons	100.00	100.00	100.00
Circle Solitaire	100.00	100.00	100.00
Hamilton	89.17	84.05	63.65
Hanoi	80.00	60.00	60.00
Hunter	87.00	75.12	84.60
Knights Tour	89.70	69.17	58.65
Lights Out	0.00	0.00	0.00
Rubik's Cube	0.00	0.00	14.85
Untwisty Corridor	6.00	6.00	100.00
Average	**55.19**	**49.53**	**58.18**

is more meaningful than it actually looks like. **Buttons** and **Circle Solitaire** turned out to be games simple enough that all players were able to solve them. **Knights Tour** and **Untwisty Corridor** can be fully solved by all three players under $120/30$ time constraints, however the advantage of DMP could be observed with shorter clocks. **Hunter** can only be solved partially and in this game DMP gains advantage under the shortest times. In **Hanoi** the proposed enhancements enable DMP to find the second highest score - 80 - in the $2/1$ clocks setup whereas BMP and CP can only reach the 60 points. However, when there is more time available, only CP is capable to solve the game regularly. CP performs better in Rubik's Cube too, which is the only game in which our player performs really poorly.

Generally speaking, we recommend applying a few low-cost techniques which proved beneficial in our tests, i.e. memorization of a path leading to the best solution found so far, using the maximal instead of the average score in the UCT formula and taking advantage of the transposition tables.

The first of the above-mentioned improvements is obviously beneficial and there are no disadvantages related to its using.

The maximal score performed better in our experimental comparison most probably due to the fact that in single-player games most of the simulations end with the score of zero, so the average values are also distributed near zero. Not only this may cause the UCT algorithm to be numerically unstable but can also make exploration to exploitation ratio hard to balance.

The only drawback of using transposition tables enhancement is a computational overhead of matching the transposed states. However, in GGP, the inference required to perform simulations is already slow, so additional burden is relatively insignificant. All of the enhancements can work together in any combination.

The proposed simulation strategies work in a combination and each strategy has some chance of finding a solution. The more strategies used by the player, the higher probability that some strategy will find a positive score in a given game. However, as our simulation results show, using too many strategies is usually not beneficial since relatively more time is allotted (wasted) for the worse ones.

In general, our approach works well in games for which the Monte-Carlo simulation-based approach is well suited. Introduction of the proposed strategies helps the MCTS process find the solutions much faster. The most favorable case seems to be a game having multiple progressive goals, where achieving consecutive goals drives the player monotonically towards the best solution. Our approach is also well suited for games in which it is easy to repeat a state, but the correct strategy is to avoid repeating the states often (e.g. the Eight Puzzle).

In summary, MINI-Player's version specialized for single-player games accomplishes better average scores than its competitors under longer time settings, whereas in the scenario where the START and PLAY clocks are initialized with very small values, CadiaPlayer performs slightly better.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2015.2391232, IEEE Transactions on Computational Intelligence and AI in Games

VIII. Related Work

To the best of our knowledge, there is not much related work dedicated to single-player games in GGP.

The most notable approach is Nested Monte Carlo Search (NMCS) [41] which was first introduced to GGP by the authors of Ary program [23]. The basic idea of NMCS is to perform a principal playout with a bias on the selection of each move based on the results of MCTS. The NMCS does not operate on average scores in a tree but instead recurrently calls itself for subsequent levels. Once it visits terminal nodes, the best score found in terminal nodes is back-propagated and the max function is applied in each node. Searches at the highest level are repeatedly performed until the thinking time elapses. As stated in [41], the complexity of NMCS search at level n equals $O(a^n h^{n+1})$, where a denotes the average branching factor and h denotes the tree height.

The authors of Gamer [42], which is another GGP agent, propose a mechanism of solving games based on symbolic search and binary decision diagrams. However, many games are too complex to be completely solved in a reasonable amount of time. Moreover, the approach requires a complete instantiation of all formulas in the GDL input which is often infeasible. The authors show that promising results are achieved for some of the puzzles, such as Hanoi or Lights Out.

Another work relies on applying the Answer Set Programming (ASP) technique [43]. ASP is used for the purpose of performing depth-restricted, complete forward search from the current position so as to find a solution in the end-game phase. One of the key ASP features is the requirement for mapping a GDL specification onto the ASP program, which is not feasible for every game. In [43] the author reports 4 games out of 13 tested which failed to be converted. Two games out of the remaining 9, Eight Puzzle and KnightsTour, were also used by us and the average solving times are similar. The ASP techniques might work better that the UCT method for simpler games in terms of the GDL description length and complexity of rule dependencies. Answer Set Programming is also useful for performing a complete search in the endgame, when the remaining tree complexity is relatively low, to find the perfect solution. Our method is based on exploration and simulation and most probably better suited for games in which a solution must be found with a bit of luck (is generally hard to find). Furthermore, the simulation strategies that we employ speed-up the convergence of the MCTS algorithm.

All the above-mentioned works differ from our approach in a sense that they either rely on historical scores in terminal states or, optimized, but still *brute-force*, complete state-space search.

IX. Conclusion

In this paper a method of adapting a UCT-based General Game Playing agent to single-player games is presented. The proposed enhancements include: (1) six variants of the so-called Exploration Strategy, which is a light-weight policy used to guide Monte Carlo simulations, (2) an optimized UCT formula, (3) a custom way of implementing state transpositions and (4) a modified tree expansion scheme, among which the enhancements (1) and (3) are entirely novel contributions.

A standard (baseline) version of our GGP agent - MiNI-Player - and the one with the proposed enhancements were compared based on the results in 10 puzzles, each solved 100 times, under 3 different time settings. Out of 30 experiments (10 games x 3 clocks settings) the puzzle-dedicated player dominated the baseline version by achieving a higher average score in 15 trials and being tied for the score in the remaining 15 cases. In terms of an average score, our puzzle-dedicated player is also better than CadiaPlayer but this result may be affected by external factors, e.g. inference engine performance.

It is worth underlying that multi-player games generally test adversarial or cooperative skills of the playing agents, while puzzles can be rather regarded as constrained search or optimization problems. The results proved that due to the above-stated fundamental difference, single-player games deserve separate attention.

Our current focus is on investigating the possibilities of combining our solution with the ideas presented in section VIII, since most of the enhancements are independent from each other and potentially may lead to better results when applied simultaneously. We also work on widening the repertoire of simulation strategies used by MiNI-Player in the case of both puzzles and multi-player games. One of the viable options is the use of an explicit evaluation function, devised in another GGP approach that we developed [44], [45], combined with the soft-max policy of move selection.

Our long-term goal is to bring in some pattern-based enhancements to the simulation phase, therefore making the solution a more cognitively plausible and human-like approach [46].

Acknowledgment

This research was supported by the National Science Centre in Poland grant number DEC-2012/07/B/ST6/01527 and Multi-plAtform Game Innovation Centre (MAGIC), funded by the Singapore National Research Foundation under its IDM Futures Funding Initiative and administered by the Interactive & Digital Media Programme Office, Media Development Authority.

M. Świechowski was also supported by the Foundation for Polish Science under International Projects in Intelligent Computing (MPD) and The European Union within the Innovative Economy Operational Programme and European Regional Development Fund.

References

[1] A. L. Samuel, "Some studies in Machine Learning using the game of Checkers," *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 210–229, Jul. 1959. [Online]. Available: http://dx.doi.org/10.1147/rd.33.0210

[2] C. E. Shannon, "XXII. Programming a computer for playing Chess," *Philosophical Magazine (Series 7)*, vol. 41, no. 314, pp. 256–275, 1950.

[3] M. Buro, "The Evolution of Strong Othello Programs," in *Entertainment Computing*, ser. The International Federation for Information Processing, R. Nakatsu and J. Hoshino, Eds. Springer US, 2003, vol. 112, pp. 81–88.

[4] G. Tesauro, "TD-Gammon, a self-teaching Backgammon program, achieves master-level play," *Neural Computations*, vol. 6, no. 2, pp. 215–219, Mar. 1994. [Online]. Available: http://dx.doi.org/10.1162/neco.1994.6.2.215

[5] B. Sheppard, "World-championship-caliber Scrabble," *Artificial Intelligence*, vol. 134, pp. 241–275, 2002.

[6] S. Samothrakis, D. Robles, and S. Lucas, "Fast Approximate Max-n Monte Carlo Tree Search for Ms Pac-Man," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 2, pp. 142–154, 2011.

[7] B. Brügmann, "Monte Carlo Go," 1993. [Online]. Available: ftp://ftp.cgl.ucsf.edu/pub/pett/go/ladder/mcgo.ps

[8] O. Syed and A. Syed, *Arimaa - A New Game Designed to be Difficult for Computers*. Institute for Knowledge and Agent Technology, 2003, vol. 26, no. 2.

[9] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen, "Checkers is solved," *Science*, vol. 317, no. 5844, pp. 1518–1522, 2007.

[10] V. Allis, "A knowledge-based approach of Connect-Four," Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands, 1988.

[11] F.-H. Hsu, *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton, NJ, USA: Princeton University Press, 2002.

[12] M. Buro, "The Othello match of the year: Takeshi Murakami vs. Logistello," *ICCA Journal*, vol. 20, no. 3, pp. 189–193, 1997.

[13] M. Świechowski and J. Mańdziuk, "Specialized vs. multi-game approaches to AI in games," in *Intelligent Systems'2014*, ser. Advances in Intelligent Systems and Computing, P. Angelov, K. Atanassov, L. Doukovska, M. Hadjiski, V. Jotsov, J. Kacprzyk, N. Kasabov, S. Sotirov, E. Szmidt, and S. Zadrożny, Eds. Springer International Publishing, 2015, vol. 322, pp. 243–254.

[14] J. Mańdziuk, *Knowledge-Free and Learning-Based Methods in Intelligenet Game Playing*, ser. Studies in Computational Intelligence. Berlin, Heidelberg: Springer-Verlag, 2010, vol. 276.

[15] J. Pitrat, "Realization of a general game-playing program," in *IFIP Congress (2)*, 1968, pp. 1570–1574.

[16] M. Gherrity, "A game-learning machine," Ph.D. dissertation, University of California at San Diego, La Jolla, CA, USA, 1993, UMI Order No. GAX94-14755.

[17] S. L. Epstein, "Toward an ideal trainer," *Machine Learning*, vol. 15, no. 3, pp. 251–277, 1994.

[18] B. Pell, "METAGAME: A new challenge for games and learning," in *Programming in Artificial Intellegence: The Third Computer Olympiad. Ellis Horwood*. Ellis Horwood Limited, 1992, pp. 237–251.

[19] M. R. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazine*, vol. 26, no. 2, pp. 62–72, 2005. [Online]. Available: http://games.stanford.edu/competition/misc/aaai.pdf

[20] J. Clune, "Heuristic Evaluation Functions for General Game Playing," in *AAAI*. AAAI Press, 2007, pp. 1134–1139. [Online]. Available: http://www.aaai.org/Papers/AAAI/2007/AAAI07-180.pdf

[21] S. Schiffel and M. Thielscher, "Fluxplayer: A Successful General Game Player," in *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*. AAAI Press, 2007, pp. 1191–1196.

[22] H. Finnsson and Y. Björnsson, "Simulation-based approach to General Game Playing," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*. Chicago, IL: AAAI Press, 2008, pp. 259–264.

[23] J. Méhat and T. Cazenave, "Ary, a General Game Playing program," in *Board Games Studies Colloquium*, Paris, 2010.

[24] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proceedings of the 17th European conference on Machine Learning*, ser. ECML'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 282–293.

[25] M. Świechowski and J. Mańdziuk, "Self-Adaptation of Playing Strategies in General Game Playing," *IEEE Transactions on Computational Intelligence and AI in Games*, 2014, Accepted for Publication (2014). Available in Early Access. DOI = 10.1109/TCIAIG.2013.2275163.

[26] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General Game Playing: Game Description Language specification," 2008. [Online]. Available: http://games.stanford.edu/readings/gdl_spec.pdf

[27] "Game Definition Language," 2013. [Online]. Available: http://games.stanford.edu/games/gdl.html

[28] C. Ward and P. Cowling, "Monte Carlo search applied to card selection in Magic: The Gathering," in *IEEE Symposium on Computational Intelligence and Games, 2009. CIG 2009.*, 2009, pp. 9–16.

[29] G. Chaslot, M. H. M. Winands, I. Szita, and H. J. van den Herik, "Cross-Entropy for Monte-Carlo Tree Search," *ICGA Journal*, no. 3, pp. 145–156, 2008.

[30] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, March 2012.

[31] M. Świechowski and J. Mańdziuk, "Fast Interpreter for Logical Reasoning in General Game Playing," *Journal of Logic and Computation*, 2014, DOI = 10.1093/logcom/exu058. [Online]. Available: http://logcom.oxfordjournals.org/content/early/2014/10/01/logcom.exu058

[32] J. Mańdziuk and M. Świechowski, "Generic Heuristic Approach to General Game Playing," in *SOFSEM*, ser. Lecture Notes in Computer Science, G. G. S. K. M. Bieliková, G. Friedrich and G. Turán, Eds., vol. 7147. Springer, 2012, pp. 649–660.

[33] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, May 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1013689704352

[34] dresden.de, "Dresden online games repository," 2010. [Online]. Available: http://130.208.241.192/ggpserver/public/show_games.jsp/

[35] ggp.org, "Tiltyard online games repository," 2010. [Online]. Available: http://www.ggp.org/view/tiltyard/games/

[36] Stanford, "Stanford gamemaster online repository," 2012. [Online]. Available: http://gamemaster.stanford.edu/showgames

[37] J. Méhat and T. Cazenave, "A Parallel General Game Player," *KI*, vol. 25, no. 1, pp. 43–47, 2011. [Online]. Available: http://www.springerlink.com/content/a83gt77455j377k8/

[38] A. Kishimoto and J. Schaeffer, "Transposition Table Driven Work Scheduling in Distributed Game-Tree Search," in *Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, ser. AI '02. London, UK, UK: Springer-Verlag, 2002, pp. 56–68.

[39] A. L. Zobrist, "A new hashing method with application for game playing," *ICCA journal*, vol. 13, no. 2, pp. 69–73, 1970.

[40] Cadia, "CadiaPlayer WWW site," 2013. [Online]. Available: http://cadia.ru.is/wiki/public:cadiaplayer:main/

[41] J. Méhat and T. Cazenave, "Combining UCT and Nested Monte Carlo Search for Single-Player General Game Playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 271–277, 2010. [Online]. Available: http://www.lamsade.dauphine.fr/ cazenave/papers/ggp2009.pdf

[42] P. Kissmann and S. Edelkamp, "Gamer, a General Game Playing Agent," *KI*, vol. 25, no. 1, pp. 49–52, 2011. [Online]. Available: http://www.springerlink.com/content/y646p6603131k581/

[43] M. Thielscher, "Answer Set Programming for Single-Player Games in General Game Playing," in *Proceedings of the International Conference on Logic Programming (ICLP)*. Springer, 2009. [Online]. Available: http://www.cse.unsw.edu.au/ mit/Papers/ICLP09.pdf

[44] K. Walędzik and J. Mańdziuk, "Multigame playing by means of UCT enhanced with automatically generated evaluation functions," in *AGI11*,

ser. Lecture Notes in Artificial Intelligence, vol. 6830. Springer, 2011, pp. 327–332.

[45] ——, "An Automatically-Generated Evaluation Function in General Game Playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 3, pp. 258–270, 2014.

[46] J. Mańdziuk, "Towards cognitively plausible game playing systems," *IEEE Computational Intelligence Magazine*, vol. 6, no. 2, pp. 38–51, 2011.

Maciej Świechowski received the B.Sc. and the M.Sc. in computer science from the Warsaw University of Technology, Warsaw, Poland, in 2009. He is currently pursuing the Ph.D. in the Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland under the International Projects in Intelligent Computing Programme. His PhD thesis is planned for defense in 2015. His research interests include artificial intelligence in games, general game playing, graph theory, computational intelligence, and computer graphics.
He has participated in the 2012-2014 International General Game Playing Competitions. He has also acquired practical programming experience in the mobile video games industry.

Jacek Mańdziuk Ph.D., D.Sc., received M.Sc. (Honors) and Ph.D. in Applied Mathematics from the Warsaw University of Technology, Poland in 1989 and 1993, resp., D.Sc. degree in Computer Science from the Polish Academy of Sciences in 2000 and the title of Professor Titular in 2011.

He is an Associate Professor at the Warsaw University of Technology and Head of the Division of Artificial Intelligence and Computational Methods. His current research interests include application of Computational Intelligence methods to game playing, dynamic optimization problems, bioinformatics, financial modeling and development of general-purpose human-like learning methods.

He is an Associate Editor of the IEEE Transactions on Computational Intelligence and AI in Games and an Editorial Board Member of the International Journal On Advances in Intelligent Systems. He is a recipient of the Fulbright Senior Advanced Research Award.

Yew-Soon Ong received a PhD degree on Artificial Intelligence in complex design from the Computational Engineering and Design Center, University of Southampton, UK in 2003. His current research interest in computational intelligence spans across memetic computation, evolutionary design, machine learning and Big data.

He is the founding Technical Editor-in-Chief of Memetic Computing Journal, founding Chief Editor of the Springer book series on studies in adaptation, learning, and optimization, Associate Editor of the IEEE Transactions on Evolutionary Computation, the IEEE Transactions on Neural Networks & Learning Systems, IEEE Computational Intelligence Magazine, IEEE Transactions on Cybernetics, Soft Computing, International Journal of System Sciences and others.