Fuegito: an Educational Software Package for Game Tree Search

Colin Hunt and Martin Müller
University of Alberta
Edmonton, Canada
{colin,mmueller}@ualberta.ca

No Institute Given

Abstract. Fuegito is an educational software package for learning about programming two player games. The package provides a simple, yet flexible and extensible framework which allows students to study the core search algorithms of computer game-playing, and extend them easily in projects.
The current version of Fuegito supports alphabeta and Monte Carlo Tree Search as main game tree search methods, as well as a number of simple playing algorithms.

1 Introduction

From its very beginnings, the history of computing science has been closely intertwined with the development of both game theory and computer game-playing, leading to programs that play classical two player games such as chess, checkers and Go.

Successes such as IBM's Deep Blue, which won a match against world chess champion Garry Kasparov, have fascinated a broad audience. Technical innovations such as Zobrist hashing or massively parallel search have found their way into mainstream computing science.

Today, millions of people enjoy programs which play board games on devices ranging from cell phones to multicore computers. In computing science education, the basic techniques such as alphabeta game tree search are taught in most introductory undergraduate courses on Artificial Intelligence, and are covered in standard textbooks. Specialized courses that teach a deeper knowledge of game-playing and game tree search algorithms are offered at universities such as UC Berkeley, University of Alberta and University of Maastricht.

A sample curriculum for such a course, Cmput 655 taught by Müller at University of Alberta, includes the following topics: Minimax search, alphabeta, board representation, solving a game vs playing well with heuristic search, proof trees, heuristic evaluation functions, refinements such as hashing, move ordering, pruning, search extensions, parallel game tree search, proof number search and refinements, AO*, Monte Carlo simulations and Monte Carlo Tree Search.

One issue with teaching a games-related course at both undergraduate and graduate levels is which software to use for developing games.

1.1 Software for Game Tree Search: Learning by Doing

It is natural to use game-playing software for teaching game programming. The number of publicly available codes for game-playing is immense, ranging from toy projects and sample code distributed with textbooks to serious, high-performance code. However, only a small fraction of these codes is designed for educational use. This section reviews a small number of relevant projects which have played a part in motivating the current work.

FUEGITO is designed to be a "little brother" to the FUEGO framework [6]. Where FUEGO provides state of the art implementations of game tree search algorithms such as alphabeta search and (especially) Monte Carlo Tree Search, FUEGITO provides a much smaller, simpler framework with implementations of such algorithms for educational use.

The second author's motivation for this work goes back to work done in the late 1980's by Kierulf and his colleagues at UNC Charlotte and ETH Zurich. SMART GAME BOARD [11] was a multi-year project to develop a sophisticated game-independent electronic game board and user interface. SMART GAME BOARD included a game-independent programming library called SMARTGAME which supported the development of several state of the art game-playing programs. SMART GAME BOARD was used for teaching game programming and heuristic search courses in the late 1980's and 1990's at ETH Zurich, Switzerland. As the main focus of a PhD thesis, SMART GAME BOARD was quite a large, complex program. Therefore, for educational purposes a much smaller SIMPLE GAME BOARD was developed at ETH Zurich by Kierulf. This unpublished program was used exclusively for teaching courses.

Another closely related work is Garcia's GAMESMAN. For many years, Dan Garcia at UC Berkeley has run an undergraduate game research and development group called GamesCrafters. Their GAMESMAN software [8] has grown into a massive project including a very large number of student projects implementing a large number of games on many different programming environments.

CGSUITE [13] is a full-featured, integrated suite of tools specialized for combinatorial game theory (CGT). It provides operations for disjunctive sums of games. It has been used in many courses and student projects focused on CGT.

An example of a game-specific software with educational focus is Drake's OREGO [3], which has been used extensively in undergraduate student projects focused on Monte Carlo Tree Search and the game of Go. Today, many high-performance game-playing engines are available as open source. While these provide fascinating reading for advanced students, their focus in not on CS education.

2 The Design and Implementation of Fuegito

FUEGITO is designed from the ground up to be a simple, universal framework for implementing games and search algorithms. Specifically, FUEGITO is concerned with two-player, deterministic games that use perfect information and game-tree

search algorithms. This section describes the design goals of FUEGITO as well as the framework's main functionality.

2.1 Design Goals

The principal design goals for FUEGITO are as follows:

- **Provide a framework for two-player deterministic games with perfect information.** FUEGITO supports any game that fits these criteria with a library of simple standard game-playing algorithms and players. FUEGITO implements several generic game-tree search algorithms that can operate on any such game, and provides generic players that use such algorithms to play these games.
- **Keep similarity with Fuego.** FUEGITO was conceived in part to act as an analogue with FUEGO [6], a state of the art game framework that is more than an order of magnitude more complex. To this end, FUEGITO was designed with a similar overall structure to its modules and classes. Same or similar names have been retained where possible, in order to allow an easier transition for students who wish to step up to developing for the more complex FUEGO later. Literally, fuego means "fire" in Spanish while fuegito means "little fire".
- **Be easy to understand and simple to use.** While FUEGITO was inspired by FUEGO, FUEGITO is primarily concerned with simplicity in its design and ease of use. This means that some of the more powerful features of FUEGO and some of the performance are sacrificed to keep things simple. This keeps FUEGITO more accessible: Students who are just starting with game programming can study and extend the existing framework in small steps, and program (parts of) a new game or search algorithm quickly and easily.
- **Be flexible and highly extensible.** FUEGITO users should be able to implement any new game, any new player, and/or any new search algorithm and use them in any combination with the existing games, players, and searches.
- **Provide a framework for teaching game programming.** The simplicity and extensibility of FUEGITO should make it an ideal framework for teaching purposes. Anyone with some introductory C++ knowledge, or who is currently learning C++, should find FUEGITO accessible. This makes FUEGITO ideal as a teaching tool in a game-programming course concerned with the fundamentals. Its extensibility means it will also be relevant in a more advanced course. An instructor or teaching assistant can quickly program the specific games and algorithms that their course is concerned with; they could also assign such tasks to the students as an exercise.
- **Make it easy to combine different games with different game-playing algorithms.** FUEGITO defines abstract interfaces between all its games, players, and search algorithms. This ensures that specific implementations can work with each other uniformly. New games can be played with

any player, new players can play any game, etc. All specific functionality is encapsulated in derived classes that use a public interface. Game-specific functionality in algorithms can be provided by the game classes themselves, freeing the algorithms from those details.
- **Provide reasonable default implementations while allowing game-specific overrides.** FUEGITO is not just a hollow framework. It provides a number of concrete implementations of games, search methods, and players to enable small but meaningful student projects. For example, a newly implemented game can use existing generic players and game-playing algorithms. As another example, implementing a game-specific alphabeta enhancement with Clobber requires only a small function overriding a function in the alphabeta code. Still, it is possible to write new games, searches, or players completely from scratch, and plug them into the framework.
- **Create a standalone open source program written entirely in standard C++.** FUEGITO is open source and written entirely in C++, using nothing but the Standard Library. This makes FUEGITO highly portable across multiple platforms with no external dependencies. While FUEGO also strives for portability, it does depend on the Boost library.
- **Create a rapid prototyping tool for designing game playing algorithms.** Because of its small size, it is easy to try out new designs and algorithms in FUEGITO. If they work well, they can then be ported to a more complex program such as FUEGO.

2.2 The Fuegito Implementation

FUEGITO is an open source project. A detailed manual is available [10]. The homepage is `http://sourceforge.net/projects/fuegito`.

Building Fuegito from Sourceforge FUEGITO uses the popular Subversion source control management system. The program can be built on any popular platform with UNIX-style command line support, such as Linux, Mac OS X, or Windows with tools such as Cygwin. The current version of the FUEGITO source code can be downloaded from the SourceForge repository with the command:

```
svn checkout svn://svn.code.sf.net/p/fuegito/code/trunk fuegito
```

Typing `make` in the `fuegito` directory then builds the `fuegito` program.

FUEGITO can be used either in pure text mode at a terminal, or use text-based communication to a graphical user interface.

Text-based Input and Output via GTP Like FUEGO and many other popular game-playing and user interface programs, FUEGITO supports the Go Text Protocol, GTP [7][1]. The list of supported commands is provided in the FUEGITO manual [10]. FUEGITO can also be configured with command-line arguments.

[1] Despite its name, this protocol can be used for any game.

Running Fuegito from the Terminal FUEGITO can be started from the terminal for a specific game with the following command:

```
fuegito -g <game_name>
```

Optional command-line arguments are:

```
-w <white_player_name> <optional_white_search_method>
-b <black_player_name> <optional_black_search_method>
```

For example, to start FUEGITO playing Clobber with average player as white and search player as black using alphabeta search:

```
fuegito -g clobber -w average -b search alphabeta
```

Players can also be specified and changed later, as shown in the sample run below. Once FUEGITO is running, it is ready to accept GTP commands.

A Sample Run The sample run starts FUEGITO from the command line for the game of Tic Tac Toe, sets up a match between an alphabeta and a Monte Carlo player, and shows how to generate moves and determine the winner.

```
./FuegitoMain -g "tictactoe"
```

The welcome message is displayed and FUEGITO is ready to accept GTP commands. First, set the white and black players to simple Monte Carlo and alphabeta respectively. In the log below, the `>` prompt indicates a user command.

```
>setplayer white search simple_montecarlo
>setplayer black search alphabeta
```

Now, specify the number of simulations per move that Monte Carlo will run. In GTP, engine responses start with `=`.

```
>white_nu_simulations 1000
= white_nu_simulations set to 1000
```

Set the search depth for the alphabeta search:

```
>black_depth 99
= Depth set to 99
```

Generate and play moves for both sides in turn:

```
>genmove white
= b2
>genmove black
= a1
...
```

When the game ends, check who won the game:

```
>getwinner
= black
```

Standard GTP Commands and Extensions Most standard GTP commands [7] are supported by FUEGITO. An extensive set of custom commands deals with listing and setting the players, search methods and supported games, among others. Furthermore, the program supports user-defined GTP commands. Details of using and creating GTP commands are described in the manual [10].

2.3 Overview of Main Classes and Their Interaction

The FUEGITO framework consists of three main components. Following the naming convention in the game-independent `SmartGame` modules of FUEGO, the base classes for these components are named `SgGame`, `SgSearch`, and `SgPlayer`. Each of these base classes defines an interface through which other classes interact with it. The complete interface for each component consists of public methods of the primary abstract base class, global constants, and further custom defined classes and types. A high-level overview is given here. Details are available in the user manual [10].

The dependencies between the main components are as follows: `SgGame` has no dependencies, `SgSearch` depends on `SgGame`, and `SgPlayer` depends on both `SgGame` and `SgSearch`.

`SgGame` is an abstract base class which defines the interface for representing a game in FUEGITO. Classes implementing specific games can either derive directly from `SgGame`, or from the utility class `SgGridBasedGame` discussed below.

`SgSearch` is an abstract class that defines the interface for a search algorithm. The term "search" is interpreted broadly here and includes some very basic move selection methods. Specific searches can inherit directly from `SgSearch`. `SgSearch` contains a pointer to an external `SgGame`-derived object that it performs the search on. FUEGITO provides a considerable number of specific implementations of different search methods.

`SgPlayer` is an abstract class that defines the interface for a player. Specific players can inherit directly from this class. `SgPlayer` contains a copy of the game (an instance of a `SgGame`-derived class) which is being played. Players can optionally use a search object (of a `SgSearch` subclass) to select a move to play. For flexibility, such search objects are created for each move decision. In this way, in FUEGITO any search can be used with any game at any time, any game can be used with any player, and any player can use any search.

2.4 Specializations of SgGame: SgGridBasedGame and Implemented Sample Games

`SgGridBasedGame` is an abstract utility subclass of `SgGame`. It provides data and methods that are common to the many games which are played on a grid, such as Go or chess. A specific grid-based game class should inherit from `SgGridBasedGame`.

FUEGITO provides sample implementations of two grid-based games: TIC TAC TOE and CLOBBER.

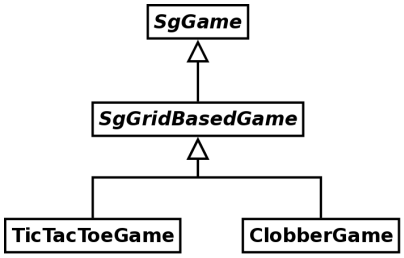

Fig. 1. Game classes inheritance diagram.

Tic Tac Toe demonstrates implementing a game with the FUEGITO framework and using the provided search algorithms. Its simplicity makes the game a good initial testing ground for implementing search algorithms, since correct behavior can be verified by hand.

Clobber was created in 2001 by Michael H. Albert, J.P. Grossman and Richard Nowakowski. It is a relatively new combinatorial game with complex gameplay, about which comparatively little is known [1]. Clobber is implemented in FUEGITO to provide an example of a more complex game with which to experiment.

2.5 Search Algorithms

FUEGITO provides implementations of several search algorithms, as shown in Figure 2: simple "searches" implement basic beginner strategies. Standard alphabeta search, Monte Carlo Tree Search (MCTS) including UCT, and several simple tree-less Monte Carlo search methods are implemented.

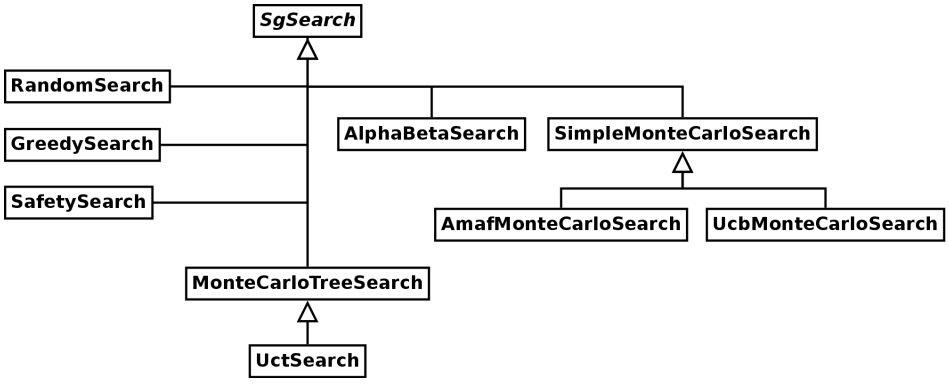

Fig. 2. Search classes inheritance diagram.

Simple "Searches" The three simple generic algorithms use basic logic to generate a move. They can be used in a simple player, or as a component of a more complex algorithm, such as move selection in Monte Carlo simulations. `RandomSearch` selects a move uniformly at random from among all legal moves. `GreedySearch` looks for a move that gives an immediate win. `SafetySearch` tries to find a move that will prevent the opponent from winning immediately on their next move.

Alphabeta Search Alphabeta search is a classic game-tree search method [12] that has been used in games such as checkers and chess for decades. FUEGITO implements the negamax version of Alphabeta search with some enhancements [10].

Simulation Policy for Monte Carlo Search A simulation policy controls the moves played in Monte Carlo simulations. A `PlayGame` method performs one complete simulation until the end of the game and returns the game result. `GenerateMove` generates a single move in a playout. FUEGITO'S default simulation policy selects uniformly at random from among all legal moves. It can be replaced by a custom policy which uses a different move generator, for example one that tries to avoid blunders.

Monte Carlo Tree Search FUEGITO'S support for MCTS is layered, similarly to the approach in FUEGO. A pair of classes, `SgUctNode` and `SgUctTree`, provide a game-independent implementation of a Monte Carlo search tree. A `MonteCarloTreeSearch` class provides an implementation of MCTS that builds a tree, using a user-supplied policy for playouts. It also acts as a base class that can be extended to implement MCTS variations such as UCT (see below). `MonteCarloTreeSearch` inherits from `SgSearch` and can be freely swapped with other search methods within FUEGITO. A derived class `UctSearch` implements the UCB1 selection policy. Unlike FUEGO, all searches are single-threaded, which greatly reduces the complexity of the software.

Three Simple Monte Carlo Algorithms FUEGITO implements three simple, tree-less methods which estimate the strength of each move by the observed winning probability of random simulations starting with that move. After a user-specified number of such simulations, a move with highest win rate is selected.

FUEGITO'S default algorithm selects a first move in round-robin fashion, so that each legal move gets an (almost) equal number of tries. The uniformly random simulation policy is used to choose all remaining moves in a playout. The win rate of all first moves is tracked, and finally, a "best" move is chosen.

The AMAF (All-Moves-As-First) algorithm [2] accumulates move statistics for all moves played during the playout, not just initial moves. A major advantage of AMAF is that statistics are accumulated at a much faster rate than in

basic Monte Carlo. FUEGITO's implementation chooses its "best" move to play by maximizing a linear combination of each move's winrate and its AMAF value.

The third simple Monte Carlo algorithm in FUEGITO is UCB [2], an upper confidence bound selection strategy for determining which initial move to explore in each simulation. The most explored move is selected finally.

2.6 Specific Players

Three sample generic players were implemented to illustrate how to combine classes from the FUEGITO toolbox to create game players. `AveragePlayer` illustrates combining multiple search techniques. It calls three simple "searches" as move generators in turn, and returns the first legal move found by any of them: `GreedySearch`, `SafetySearch` and finally `RandomSearch`. `MixedMcAbPlayer` combines Monte Carlo and alphabeta search. It starts each game with Monte Carlo search and records the remaining depth of the game tree. As soon as the estimated depth of the rest of the game is shallow enough, it will try to solve the game with alphabeta. If that fails, it reverts to Monte Carlo for one move. `SearchPlayer` is a generic player that can use any search method which implements the `SgSearch` interface. This eliminates the need to create a separate player for each different game tree search method.

2.7 Utility Classes for Game Tree Search

FUEGITO provides a number of utility classes which are useful for writing game tree search code. Many of them are closely related to their counterparts in FUEGO. For a detailed description please see the manual [10].

The `SgMoveTimer` class provides a simple timer. `SgHashTable`, `SgHashEntry`, and `ZobristNumbers` implement a Zobrist style hash table as in FUEGO [5]. The `SgHashStatistics` structure is used to gather information about hash table usage. It records data such as the number of stores, lookups, hash collisions on store, and size of the table. The `SgGridBasedGameRecord` utility class sets up a typical game-playing or testing scenario. This class can be used by a user interface to simplify running a complete game. A `GtpInterface` utility class translates back and forth between GTP and FUEGITO.

3 Evaluation of the Fuegito Framework

Evaluation consists of two parts. First, a review of the functionality achieved and how well it achieves its design goals as an educational tool. Second, a comparison with other similar software frameworks.

3.1 Using and Extending Fuegito

FUEGITO is designed to be extended in several ways. Ideas for possible student projects using the FUEGITO code base include:

– Implement a new grid-based board game, maybe one of the hundreds of games described on `http://boardgamegeek.com`.
– Implement a game not based on a grid, such as a card game.
– Implement a search method such as df-pn or a new variant of Monte Carlo Tree Search.
– Implement a search enhancement such as ProbCut or Realization Probability Search for alphabeta, or smarter playouts for Monte Carlo.
– Implement a new player type, for example one with smarter full-game time management.
– Extend the framework by providing generic support for creating and using endgame databases.

From the framework point of view, the main choices for extending FUEGITO are implementing a new game, search algorithm or player, or improving existing search algorithms by extending or customizing them. FUEGITO provides many "hooks" for such improvements.

3.2 Evaluating the Fuegito Code Base

The first main comparison is between FUEGITO on one side, and FUEGO and GAMESMAN on the other side. The second main comparison is between using a game framework and programming a game from scratch.

At about 7000 lines of code, the complexity of the FUEGITO code base is far less than either FUEGO or GAMESMAN [8]. FUEGO has close to 100,000 lines of code, while GAMESMAN, as mentioned above, is immense since it contains hundreds of student projects written for many different platforms. As of SVN revision 6923, the `gamescrafters` project on `http://sourceforge.net/projects/gamescrafters/` contains 12,748 files taking up more than 600 MB of disk space.

In all three frameworks, it is easy to add a game and combine it with an existing search algorithm. However, only FUEGITO easily supports adding a new search algorithm, and applying it to existing games without extending or modifying a class hierarchy in a game-specific way. FUEGITO is designed for extensibility from the ground up, and offers many well thought out "hooks" for doing so.

When comparing FUEGITO against FUEGO, GAMESMAN, and the option of building a game from scratch, the main advantages of FUEGITO are: small code size, ease of understanding, ease of modification and extension, and ease of switching algorithms for the same game.

Advantages of FUEGO are: more functionality, such as multithreading, better support for graphical UI through its support for the `gogui-gfx` mechanism [4], and being a more mature framework which has been tested in more applications.

Compared with GAMESMAN, there are some similar goals. The effort to code a new game seems comparable [9]. However, GAMESMAN is a massive software base while FUEGITO is a small, nimble framework. Other differences include: GAMESMAN is written in plain C, while FUEGITO and FUEGO use C++. The user interface for a new game in GAMESMAN needs to be coded in

Tcl/Tk as part of the game development, while FUEGO and FUEGITO provide a flexible GTP text interface which can be attached to any user interface that supports that standard. GAMESMAN strongly focuses on solving games, while FUEGITO focuses on being an educational tool which is closely related to the full-scale FUEGO framework.

Some shared advantages of using either framework, compared to building games from scratch, are:

- The code base and examples are available for reading/studying.
- The framework is open source, so it can profit from others working on the same code base, and the results of good student projects are contributed back to the community.
- It is easier to extend a framework than to build everything - the student can focus on the most interesting parts of programming the game, or focus on a specific algorithm that is being studied. Experience in teaching a graduate level games course (CMPUT 655) shows that even most graduate students cannot build a complete game-playing system from scratch within the time constraints of a single course.
- The framework provides a good basis for student teamwork. It allows splitting up work on a larger project into individual pieces, and testing them both in isolation and in combination.
- Default implementations allow each user to choose how much to (re-)implement and how much to re-use.

4 Conclusions and Future Work

The work on FUEGITO has shown that it is possible to implement most of a simple, yet flexible framework for two-player games as an undergraduate summer project[2]. Compared with previous frameworks, FUEGITO is orders of magnitude smaller and easier to learn, while offering comparable features and improved ease of use and extendibility.

The framework is now ready for practical use, as has been shown by the sample implementations of the games Tic Tac Toe and Clobber. Next steps include:

- Use FUEGITO to teach game programming classes or summer courses at the high school, undergraduate and graduate level.
- Provide a clear transition path from FUEGITO to the full scale FUEGO framework: backport architectural choices which worked well in FUEGITO, especially the separation of game definitions and search algorithms. Document the correspondence between the classes and functions in both programs more fully, and explain the main differences.
- Implement some extra functionality, such as multithreading, games such as Go, NoGo and Amazons, and further game tree search algorithms such as Proof-number search and df-pn.

[2] The work on MCTS was completed later, as part of a course project.

Acknowledgements

The authors gratefully acknowledge support from NSERC, including an Undergraduate Student Research Award (NSERC USRA) for Hunt.

References

1. M. Albert, J. P. Grossman, R. Nowakowski, and D. Wolfe. An introduction to clobber. *Integers, Electr. J of Combinat. Number Theory*, 5(2):#A01, 12 pp., 2005.
2. C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–49, 2012.
3. P. Drake. Orego, 2003–2013. Retrieved April 10, 2013 from `https://sites.google.com/a/lclark.edu/drake/research/orego`.
4. M. Enzenberger. GoGui documentation. Chapter 8. Live Graphics, 2013. Retrieved April 10, 2013, from `http://gogui.sourceforge.net/doc/live-gfx.html`.
5. M. Enzenberger and M. Müller. Fuego homepage, 2008-2013. Retrieved April 10, 2013 from `http://fuego.sf.net`.
6. M. Enzenberger, M. Müller, B. Arneson, and R. Segal. Fuego - an open-source framework for board games and Go engine based on Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.
7. G. Farnebäck. GTP - Go Text Protocol, 2008. Retrieved April 10, 2013, from `http://www.lysator.liu.se/~gunnar/gtp`.
8. D. Garcia. GAMESMAN: A finite, two-person, perfect-information game generator, 1995. MSc thesis, University of California, Berkeley.
9. D. Garcia. Instructions for how to create a new game, 2007. Retrieved April 10, 2013, from `http://gamescrafters.berkeley.edu/makegame.php`.
10. C. Hunt. Fuegito user manual. Technical Report TR12-05, 2012. Retrieved April 10, 2013, from `https://www.cs.ualberta.ca/research/theses-publications/technical-reports/2012/TR12-05`.
11. A. Kierulf. *Smart Game Board: a Workbench for Game-Playing Programs, with Go and Othello as Case Studies*. PhD thesis, ETH Zürich, 1990.
12. D.E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6:293–326, 1975.
13. A. Siegel. Combinatorial game suite, 2004-2013. Retrieved April 10, 2013, from `http://cgsuite.sourceforge.net`.