Vision Document
Mastergoal Machine Learning Environment
Version 1.5

Submitted in partial fulfillment of the requirements of the degree of MSE

Alejandro Alliana
CIS895 – MSE Project
<table>
<thead>
<tr>
<th>Date</th>
<th>Author</th>
<th>Revision</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>06/22/2007</td>
<td>Alejandro Alliana</td>
<td>0.1</td>
<td>Initial Draft</td>
</tr>
<tr>
<td>07/03/2007</td>
<td>Alejandro Alliana</td>
<td>0.2</td>
<td>Included Dr. Gustafson Comments</td>
</tr>
<tr>
<td>07/11/2007</td>
<td>Alejandro Alliana</td>
<td>0.3</td>
<td>Added Requirements Specification.</td>
</tr>
<tr>
<td>08/09/2007</td>
<td>Alejandro Alliana</td>
<td>0.4</td>
<td>Complete revision of the document.</td>
</tr>
<tr>
<td>08/16/2007</td>
<td>Alejandro Alliana</td>
<td>1.0</td>
<td>Presentation I Baseline</td>
</tr>
<tr>
<td>08/23/2007</td>
<td>Alejandro Alliana</td>
<td>1.1</td>
<td>Modifications required by the committee: Specific functions of the UC.</td>
</tr>
<tr>
<td>08/29/2007</td>
<td>Alejandro Alliana</td>
<td>1.2</td>
<td>Final version including presentation one committee recommendations.</td>
</tr>
<tr>
<td>11/03/2007</td>
<td>Alejandro Alliana</td>
<td>1.3</td>
<td>Updates for presentation II:</td>
</tr>
<tr>
<td>11/03/2007</td>
<td>Alejandro Alliana</td>
<td>1.4</td>
<td>Final version for presentation II Baseline</td>
</tr>
<tr>
<td>11/05/2007</td>
<td>Alejandro Alliana</td>
<td>1.5</td>
<td>Corrected requirements numbers. Added SR 2.5</td>
</tr>
</tbody>
</table>
Mastergoal Machine Learning Environment

1. Project Overview ........................................................................................................ 1
   1.1. Introduction ......................................................................................................... 1
   1.2. Purpose ................................................................................................................ 1
   1.3. Goals ................................................................................................................... 2
   1.4. Background ......................................................................................................... 2
   1.5. Risks .................................................................................................................... 4
   1.6. Constraints .......................................................................................................... 4
   1.7. Quality attributes ................................................................................................. 5
   1.8. External interfaces .............................................................................................. 5
   1.9. References ........................................................................................................... 5

2. Requirements Specification ........................................................................................ 5
   2.1. External Interfaces .............................................................................................. 6
      2.1.1. User Interfaces ............................................................................................ 6
      2.1.2. Hardware Interfaces ................................................................................... 6
      2.1.3. Communication interfaces .......................................................................... 6
   2.2. Use cases ............................................................................................................. 7
      2.2.1. Use Case Create New Experiment .............................................................. 8
         2.2.1.1. Software requirement 1.1: Create Experiment (Critical) .................. 9
         2.2.1.2. Software requirement 1.1: Select Learning Technique (Critical) ...... 9
         2.2.1.3. Software requirement 1.2: Select Terms for experiment (Critical) .... 9
         2.2.1.4. Software requirement 1.3 Setup Search Algorithm (Critical) .......... 9
      2.2.2. Use Case Load experiment .........................................................................10
         2.2.2.1. Software requirement 2.1: Load experiment (critical) .......................10
         2.2.2.2. Software requirement 2.2 Save Experiment (Critical) ......................10
         2.2.2.3. Software requirement 2.3 Edit Experiment (Critical) .......................10
         2.2.2.4. Software requirement 2.4 Experiment Format (Critical) ...................10
      2.2.3. Use Case “Train Strategy” ...........................................................................10
         2.2.3.1. Software requirement 3.1: Train Strategy (Critical) .........................11
         2.2.3.2. Software requirement 3.2 Select benchmark Individual (Non Critical) 11
         2.2.3.3. Software requirement 3.3 Adding arbitrary individuals (non critical) . 11
         2.2.3.4. Software requirement 3.4: Background processes (Non Critical) ......11
         2.2.3.5. Software requirement 3.5: Rules of play (Critical) .............................11
         2.2.3.6. Software requirement 3.6: Correct search (Critical) .........................11
      2.2.4. Use Case Export strategy ...........................................................................11
         2.2.4.1. Software requirement 4.1: Export strategy (non critical) ..................12
      2.2.5. Use Case Play Game ...................................................................................12
         2.2.5.1. Software requirement 5.1: Play a game (Critical) .............................13
         2.2.5.2. Software requirement 5.2: Rules (Critical) ........................................13
      2.2.6. Use Case Explore Game ............................................................................13
         2.2.6.1. Software requirement 6.1: Explore a game (Critical) .......................13
         2.2.6.2. Software requirement 6.2: Game Format (Non critical) .....................13
         2.2.6.3. Software requirement 6.3: Save Game (Non critical) .........................13
         2.2.6.4. Software requirement 6.4: Load Game (Non critical) .......................13
Mastergoal Machine Learning Environment.

1. Project Overview

1.1. Introduction

Mastergoal© is a board game that consists of a soccer match between two opposing teams. The board consists of 13x11 squares as in figure

![Mastergoal starting board positions for level 3](image)

Figure 1. Mastergoal starting board positions for level 3

The game can be played at levels 1 through 3 in increasing complexity, each having an increasing number of tokens or pieces and slightly different rules associated with them. The starting position of a game in level 3 is shown in Figure 1.

Research in the game started with an agreement between the creator of the game, Alberto Bogliaccini and the “Universidad Católica de Asuncion”. Research has been done to the application of adversary search algorithms such as MTD-f [Pl94] to the game with moderate success [Gi04, Qu04]. The strategy of the search has been a weak link of the implementations: Ad hoc feature vectors have been included in a linear evaluation function with hand crafted weights to guide the search.

1.2. Purpose

The Mastergoal machine learning environment will be used to learn strategies for the game of Mastergoal. Machine learning techniques such as Temporal difference
learning [Su88], Genetic Algorithms, etc. will be used in combination with adversary search trees [Gh04] [Ba98] to learn strategies for Mastergoal.

Learning strategies for games requires careful preparation and documentation of the experiments, such as training with different opponents levels [Ri03]. The Mastergoal machine learning environment should help the configuration, preparation and execution of these experiments, the collection of the results, documentation of experiments and should make the experiments repeatable.

1.3. Goals

The project goal is to provide an environment to learn strategies and useful features for the game of Mastergoal and save these strategies to be used to play against human players.

1.4. Background

Traditional approaches to Board Game AI

The main approach for producing game playing Agents [Ru95] is creating an evaluation function that returns a desirability value given a board position as input. Using this function with a depth limited minimax search improves the performance by allowing the function to look ahead many moves.

These evaluation functions can be represented by linear combinations of human chosen features, neural networks that take the state of the board as input or other techniques.

Most of the progress has been accomplished by creating improved versions of the minimax search, although this technique has also had limited success when dealing with games with high branching factor.

Reinforcement learning and Temporal difference Learning

The best understood learning techniques fall into the category of supervised learning. This category is distinguished by the fact that for each input upon which the system is trained, the “correct” output is known. This allows us to measure the error and use it to train the system.

Reinforcement learning falls into the category of unsupervised learning, which differs substantially from supervised learning in that the “correct” output is not known. Hence, there is no direct measure of error; instead a scalar reward is given for the responses to a series of inputs.

Consider an agent reacting to its environment (a generalization of the two-player game scenario). Let S denote the set of all possible environment states. Time proceeds with the agent performing actions at discrete time steps t [0..T]. At time t the agent finds the environment in state s_t, and has available a set of actions A(s_t). The agent chooses an action, a_t, which takes the environment to state s_{t+1} (possibly with probability p). After a determined series of actions in the environment, perhaps when a goal has been achieved or has become impossible, the scalar reward, r_{t+1} is awarded to the agent. These rewards are often discrete, i.e.: “1” for success, “-1” for failure, and “0” otherwise.
Reinforcement learning and temporal difference learning [Sa59] have been used to learn strategies for game playing for a long time. Basic techniques of TD-Learning were invented by Arthur Samuel to make a program that could learn to play checkers. Gerald Tesauro created TD-Gammon using the TD(λ) algorithm created by Richard Sutton [Su88]. TD-Gammon uses a combination of the TD(λ) algorithm with a Neural Network used as an evaluation function, and so it did not need to use hand crafted feature vectors: TD-Gammon used the raw representation of the board as the input for the Neural Network evaluation function.

Since then the application of temporal difference to other games such as Go and Chess have been less successful. A formal argument for this is presented in [Jo97].

Extensions of TD-Learning algorithm have been studied: TD-Leaf, which instead of defining the temporal difference as the difference between two sequential board positions uses the difference between the evaluations of the principal variation (the best move discovered by a limited ply game-tree search). TD-Directed which like TD-Leaf combines game tree search with TD-Learning. TD(μ) which examines the moves made by the opponent not learning from them if they are considered not-good.

Training Data
To learn a strategy using TD-learning we need a collection of games from which to learn. Database play, Random Play, Fixed Opponent and Self play techniques have been previously used, each of them having drawbacks.

According to [Ri03] a combination of games using different opponents improves the performance of the evaluation function learned. This is because the learning algorithm can “forget” games played in the distant past.

Learning strategies.
A number of different considerations must be taken into account when learning evaluation functions:

- Size of rewards: Is the agent given a reward for winning, tying or loosing (+1, 0, -1) or a stronger reward is given on stronger wins?
- Repetitive learning: Is learning from 300,000 different game sessions equivalent to learning from 3000 different game session 100 times?
- Learning from an inverted board: If we change the colors of the teams can the agent learn two games instead of one.
- Batch learning: Instead of learning after every game, can the agent learn after a number of games without a significant loss of performance? If this is the case the learning process can be speeded up by parallelizing the game generation.

Genetic algorithms
The Genetic Algorithms (GA) approach learns by simulating evolution. Individual hypothesis are represented as string bits or symbolic expressions in the case of Genetic Programming (GP)[Mi97].

The GA approach maintains a population $P_t$ of individuals at each step $t$. The next population $P_{t+1}$ is generated from the individuals of the current population by applying operations such as mutation and crossover. At each step the individuals are
evaluated and assigned *fitness values* according to a *fitness function*. Hypothesis with higher *fitness values* are probabilistically more likely to appear in the next generation.

The Mastergoal Machine learning environment could include a GAs experimenter that can be thought of an optimization method for the evaluation functions.

### 1.5. Risks

Project risks have been identified and activities have been defined to minimize that risk:

- **Programming Language inexperience**: Due to external constraints (see next section) the programming language chosen for the project is C++. The developer has little experience in the language, so some prototypes and experiments have been made to try to minimize the risk. The project lifecycle, modularization and other techniques should also help.

- **Exploration vs. exploitation**: The exploration versus exploitation is a problem that arises with reinforcement learning when trying to decide if the agent should take actions (or follow paths) that result in a known reward or should “explore” an unknown expecting to find new high unknown high rewards states. A number of techniques are available to overcome this problem:
  - Optimistic initial values.
  - \( \varepsilon \)-greedy algorithms

- **Computational Cost of Evaluation Functions**: The goal of the project is to try to test many of them regarding their performance not only on their accuracy to predict rewards but also in the time it takes to evaluate them, and whether this trade of is convenient.

### 1.6. Constraints

The learning environment should integrate with the *Mastergoal plug-in environment* currently been developed at the “Universidad Catolica de Asuncion”. The plug-in environment is a framework that is been created to play the game.

The integration should be done at many levels:

1. There must not be any ambiguities in the rules. Any difference in the rules might result in the learning environment learning an incorrect strategy. A common rules library can be used by both the “learning environment” and the “plug-in environment”.
2. Since some level of integration is needed, a cleaner interface will result if both projects where coded in the same programming language. Since the “plug-in environment” is coded in C++, this puts a restriction on the coding language of the “learning environment”.
3. As a clarification: The “Mastergoal plug-in” and the “learning environment” should be able to run independently. The “Mastergoal Plug-in” must be able to use the output of the “learning environment”, an evaluation function, in some way.
1.7. Quality attributes

Since an error could not be detected until the framework fails to learn and then it would be really hard to spot the bug in the code. Unit testing\(^1\) should be done in every module and the resulting artifacts should be tested independently before integration testing.

1.8. External interfaces

It would be useful if the “learning environment” had an interface that allows its features to be called from the command line. These command line interfaces will allow the program to be included in scripts.

1.9. References

[Ri03] Combining TD-learning with Cascade-correlation Networks. Francois Rivest, Doina Precup.

2. Requirements Specification

In this section, the driving requirements of the project are specified.

---

\(^1\) I have researched a number of unit testing suites, the best seems to be Cppunit (http://cppunit.sourceforge.net/cppunit-wiki)
2.1. **External Interfaces**

2.1.1. User Interfaces

The program should have both, a Graphical User Interface (GUI) and a Command Line User Interface (CLUI).

The GUI will be useful to prepare, verify, run and save experiments. The experiments themselves cannot depend on the GUI and must run as background processes, since it might take days for them to finish. The MMLE should be divided into a number of sub-programs that can run from command line.

The CLUI is the user interface for the subprograms that compose the MMLE. Each subprogram must have well defined command line parameters.

2.1.2. Hardware Interfaces

There are no hardware interfaces required at this point.

2.1.3. Communication interfaces

Figure 2. shows a component view of the system. Components are divided either into the Machine Learning Environment (MMLE) or an experiment group. Each experiment group is different and handles its own GUI.
2.2. Use cases

The main use cases include: Create new experiment, setup the learning environment for a new experiment (agents, epochs, search algorithms, evaluation type, batch learning), train a strategy, save the experiment (persist for later experimentation), load experiment, and export strategy to be used for the Mastergoal plug-in environment.
2.2.1. Use Case Create New Experiment

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Create New Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors</strong></td>
<td>Experimenter</td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>Create a new experiment to train and obtain a new strategy</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>The system must be started and loaded.</td>
</tr>
<tr>
<td><strong>Scenario</strong></td>
<td>• The experimenter selects the Train Strategy option.</td>
</tr>
<tr>
<td></td>
<td>• The experimenter inputs the experiment data, searching and learning parameters:</td>
</tr>
<tr>
<td></td>
<td>o Experiment name, date, and obs. data.</td>
</tr>
<tr>
<td></td>
<td>o Search algorithm, search depth, max time.</td>
</tr>
<tr>
<td></td>
<td>o Terms to be used for the experiment.</td>
</tr>
<tr>
<td></td>
<td>o Learning technique to be used: GA, RL, etc.</td>
</tr>
<tr>
<td></td>
<td>o Learning technique specific parameters: i.e. for GAs: Population, crossover rate,</td>
</tr>
<tr>
<td></td>
<td>mutation rate, fitness function.</td>
</tr>
<tr>
<td></td>
<td>• The experimenter selects the save experiment option</td>
</tr>
<tr>
<td><strong>Post conditions</strong></td>
<td>• The system creates a new experiment.</td>
</tr>
<tr>
<td></td>
<td>• A new directory structure is created where the logs and other files will be saved.</td>
</tr>
<tr>
<td><strong>Exceptions</strong></td>
<td>• If the application had an active experiment opened with unsaved changes, the user</td>
</tr>
<tr>
<td></td>
<td>must receive information about this.</td>
</tr>
<tr>
<td></td>
<td>• A file system error might occur.</td>
</tr>
<tr>
<td><strong>Open issues</strong></td>
<td>• Experiments and results could be saved using a DBMS or to the file system.</td>
</tr>
</tbody>
</table>
2.2.1.1. **Software requirement 1.1: Create Experiment (Critical)**

The system shall provide a GUI for creating, editing and saving experiments. Different AI learning components can and probably will require different information. i.e. A GA based components will require Population, crossover rate, mutation rate, fitness function and termination criteria while a Reinforcement Learning component will require parameters such as epochs, alpha, lambda whether batch learning should be used, etc.

2.2.1.2. **Software requirement 1.1: Select Learning Technique (Critical)**

The system shall provide the user with the option to select from the available Learning techniques for the experiment. Initially only a GA learning technique will be provided.

2.2.1.3. **Software requirement 1.2: Select Terms for experiment (Critical)**

The system shall allow the user to select a subset from the available terms or features to use in the experiment.

2.2.1.4. **Software requirement 1.3 Setup Search Algorithm (Critical)**

The system shall allow the user to select different parameters to be used by the search algorithm during training. The system shall also allow the Experimenter to select from different search algorithms. Initially only one will be provided (MTDF or Alpha-Beta)

2.2.2. **Use Case Load experiment**

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Load experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Experimenter</td>
</tr>
<tr>
<td>Goal</td>
<td>Load the experiment from the persistent media.</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
<tr>
<td>Scenario</td>
<td>• The experimenter selects the train strategy option.</td>
</tr>
<tr>
<td></td>
<td>• The experimenter selects the open experiment option.</td>
</tr>
<tr>
<td></td>
<td>• The experimenter selects the experiment to load.</td>
</tr>
<tr>
<td>Post conditions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The experiment is loaded into the GUI and the status and results are displayed.</td>
</tr>
<tr>
<td>Exceptions</td>
<td>There might be a file system error.</td>
</tr>
<tr>
<td>Open issues</td>
<td></td>
</tr>
</tbody>
</table>
2.2.2.1. **Software requirement 2.1: Load experiment (critical)**

The system shall allow the user to load previously saved experiment. The properties of the loaded experiment should match those specified by the data in the saved experiment.

2.2.2.2. **Software requirement 2.2 Save Experiment (Critical)**

The system shall allow the user to save an experiment. The data saved must match the properties of the saved experiment.

2.2.2.3. **Software requirement 2.3 Edit Experiment (Critical)**

The system shall provide an interface to edit experiments once they have been loaded.

2.2.2.4. **Software requirement 2.4 Experiment Format (Critical)**

The system shall define a format to be used when saving and restoring the experiments. This format should allow extensibility to allow saving experiments with different types of learning techniques.

![Figure 4. Train Strategy Use Case](image)

2.2.3. **Use Case “Train Strategy”**

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Train Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Experimenter</td>
</tr>
<tr>
<td>Goal</td>
<td>This is the main feature of the program. It starts training the strategy using the parameters selected for an experiment. The goal is to obtain a new strategy (set of new strategies) according to the training options and the learning procedure specified by the learning components.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>• The new experiment must be created or loaded and the training options must be set.</td>
</tr>
</tbody>
</table>
| Scenario | • The experimenter selects the *start training option*  
• The program creates the directory structure to hold the experiment.  
• The program starts training with the specified arguments. |
2.2.3.1. **Software requirement 3.1: Train Strategy (Critical)**

The program shall provide a command to train a strategy from an existing experiment. Different learning components will train their strategy differently.

2.2.3.2. **Software requirement 3.2 Select benchmark Individual (Non Critical)**

The program shall provide the experimenter with an option to select an arbitrary benchmark strategy for the experiment. A default benchmark strategy should always be provided.

2.2.3.3. **Software requirement 3.3 Adding arbitrary individuals (non critical)**

The program shall provide the experimenter with the ability to add arbitrary individuals to the experiment.

2.2.3.4. **Software requirement 3.4: Background processes (Non Critical)**

Since training could take a long time, the system should provide a way to run the training process in background. In this way, the experimenter does not have to use the GUI during all the training process.

2.2.3.5. **Software requirement 3.5: Rules of play (Critical)**

The system shall enforce the rules at every game played during training.

2.2.3.6. **Software requirement 3.6: Correct search (Critical)**

The systems search algorithms should return the best move according to the evaluation criteria and the search algorithm behaviour.

### 2.2.4. Use Case Export strategy

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Export strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Experimenter</td>
</tr>
<tr>
<td>Goal</td>
<td>Export the strategy in a format that can be used by the Mastergoal Plug-in</td>
</tr>
<tr>
<td>Preconditions</td>
<td>● A trained strategy must exist saved in the file system.</td>
</tr>
</tbody>
</table>
| Scenario     | ● The experimenter selects the *export strategy option*.  
|              | ● The experimenter selects the strategy to be exported. |
The experimenter selects the location where to export the strategy.
- The experimenter selects an exporter algorithm.

**Post conditions**
- The strategy is saved into a file.

**Exceptions**

**Open issues**

---

### 2.2.4.1. Software requirement 4.1: Export strategy (non critical)

The system shall provide a mechanism to export a learned strategy to a file in external format. This will allow the strategies learned by MMLE to be used in other programs that can play the game.

---

**Figure 5. Game Use Cases**

---

### 2.2.5. Use Case Play Game

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Play Game</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors</strong></td>
<td>Experimenter</td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>Allow the experimenter to try different strategies by playing a game against them. The interface also allows games between two human players and two programs.</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>At least one trained strategy exists in the framework</td>
</tr>
</tbody>
</table>
| **Scenario** | - The experimenter selects the *Play/Explore game* option.  
- The experimenter sets up the game information.  
- The experimenter sets up the search tree information.  
- The experimenter selects the strategy.  
- The experimenter selects the start game option. |
| **Post conditions** | |
| **Exceptions** | |
| **Open issues** | |
2.2.5.1. **Software requirement 5.1: Play a game (Critical)**

The system shall provide a GUI to play games. This user interface can be used to try the learned strategies. The games can be played between Human Players and or Computer program players.

2.2.5.2. **Software requirement 5.2: Rules (Critical)**

During a game, the system shall enforce the application of the rules at all times.

### 2.2.6. Use Case Explore Game

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Explore Game</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Experimenter</td>
</tr>
<tr>
<td>Goal</td>
<td>Allow the experimenter to scroll through moves of a game or to review games played during training.</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>At least one trained strategy exists in the framework</td>
</tr>
<tr>
<td></td>
<td>The game to be explored must be opened in the program.</td>
</tr>
<tr>
<td>Scenario</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The experimenter scrolls through the moves of the game.</td>
</tr>
<tr>
<td>Post conditions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The board gets updated when selecting the previous or next move.</td>
</tr>
</tbody>
</table>

#### 2.2.6.1. **Software requirement 6.1: Explore a game (Critical)**

The system shall provide a GUI to scroll through the moves of a game

#### 2.2.6.2. **Software requirement 6.2: Game Format (Non critical)**

The system shall specify a format to save game positions and the moves of a game. This game format shall follow the traditional syntax provided by the rules of the game to specify positions and moves.

#### 2.2.6.3. **Software requirement 6.3: Save Game (Non critical)**

The system shall allow the user to save games to a file.

#### 2.2.6.4. **Software requirement 6.4: Load Game (Non critical)**

The system shall allow the user to load games from a file.