Architecture Design
Mastergoal Machine Learning Environment
Version 0.8

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

Alejandro Alliana
CIS895 – MSE Project
## Change Log

<table>
<thead>
<tr>
<th>Date</th>
<th>Author</th>
<th>Revision</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/16/2007</td>
<td>Alejandro Alliana</td>
<td>0.1</td>
<td>Initial version</td>
</tr>
<tr>
<td>10/17/2007</td>
<td>Alejandro Alliana</td>
<td>0.1</td>
<td>Initial version</td>
</tr>
<tr>
<td>10/18/2007</td>
<td>Alejandro Alliana</td>
<td>0.1</td>
<td>Initial version.</td>
</tr>
<tr>
<td>10/19/2007</td>
<td>A Alliana &amp; DAG</td>
<td>0.2</td>
<td>Revision by Dr. David A. Gustafson</td>
</tr>
<tr>
<td>10/23/2007</td>
<td>Alejandro Alliana</td>
<td>0.2</td>
<td>Added detail in the Board representation.</td>
</tr>
<tr>
<td>10/25/2007</td>
<td>Alejandro Alliana</td>
<td>0.2</td>
<td>Added Object diagrams</td>
</tr>
<tr>
<td>11/01/2007</td>
<td>Alejandro Alliana</td>
<td>0.3</td>
<td>Modified according to OCL specifications.</td>
</tr>
<tr>
<td>11/01/2007</td>
<td>Alejandro Alliana</td>
<td>0.4</td>
<td>Submitted for formal inspection.</td>
</tr>
<tr>
<td>11/04/2007</td>
<td>Alejandro Alliana</td>
<td>0.5</td>
<td>Included modifications suggested by Inspector Kenton Born</td>
</tr>
<tr>
<td>11/07/2007</td>
<td>Alejandro Alliana</td>
<td>0.6</td>
<td>Included modifications suggested by Inspector Kenton Born</td>
</tr>
<tr>
<td>11/08/2007</td>
<td>Alejandro Alliana</td>
<td>0.7</td>
<td>Included modifications suggested by Inspector Kenton Born</td>
</tr>
<tr>
<td>11/29/2007</td>
<td>Alejandro Alliana</td>
<td>0.8</td>
<td>Added figures of Presentation II.</td>
</tr>
</tbody>
</table>
1. INTRODUCTION............................................................................................................................... 1

2. PROJECTS PACKAGE AND OVERVIEW.......................................................................................... 1
   2.1. PACKAGE DEPENDENCY............................................................................................................. 1

3. MASTERGOAL SUBPROJECT .............................................................................................................. 2
   3.1. BOARD STATUS PACKAGE.......................................................................................................... 2
       3.1.1. Structure ................................................................................................................................. 2

4. MGAII SUBPROJECT ........................................................................................................................... 10
   4.1. STRUCTURE ............................................................................................................................... 10
   4.2. BEHAVIOR ................................................................................................................................. 11

5. MMLE SUBPROJECT .......................................................................................................................... 13
5.1. MODEL PACKAGE ................................................................. 13
  5.1.1. Structure ........................................................................... 13
  Class Framework:........................................................................ 13
  Class AbstractExperimentFactory:............................................ 13
  Class AbstractExperiment: ........................................................ 13
  Class GAExperiment: ............................................................... 13
  Class TerminationCriteriaManager: .......................................... 13
  Class FitnessFunctionManager: ............................................... 13

5.2. GA EXPERIMENT .............................................................. 14
  Class Population:........................................................................ 14
  Class GAData: ........................................................................... 14
  Class Individual:........................................................................ 14
  Class AbstractTerminationCriterion: ...................................... 14
  Class AbstractFitnessFunction: .............................................. 14
  Class SelectionContext: .......................................................... 15
  Class SelectionCriterion: ....................................................... 15
  Class RankSelection: .............................................................. 15
  Class FitnessProportionateSelection: ...................................... 15
  Class TournamentSelection: ................................................... 15

5.2.1. Behavior ........................................................................... 15
  Create Experiment: .................................................................... 15
  Train Strategy: .......................................................................... 16
  Compute Fitness: .................................................................... 16

5.3. USER INTERFACE ............................................................ 17
  Class FrmMain: ......................................................................... 18
  Class FrmGame: ........................................................................ 18
  Class FrmExperiment: ............................................................. 18
  Class FrmGAPopulation: ......................................................... 18
  Class ExportStrategy: ............................................................. 18
1. Introduction

This document presents the architecture of the Mastergoal Machine Learning Environment (MMLE) project. The architecture is presented at the static structural view (class diagrams, object diagrams, component diagrams) and dynamic behavior view (sequence diagrams, activity diagrams).

The use of Design Patterns\(^1\) has been encouraged where appropriate.

2. Projects Package and overview

MMLE is divided into three subprojects:

- **Mastergoal**: The Mastergoal subproject contains the basic components that represent a game, moves, teams, etc. The goal of this project is to provide a base to other projects to play the game.
- **MGAI**: The MGAI subproject contains the components that represent agents, search algorithms and traditional AI components that can be used by other projects.
- **MMLE**: The MMLE subproject contains the components that provide an environment to use machine learning techniques to learn strategies for the game of Mastergoal. This subproject uses the components of the last two projects.

The MMLE subproject is going to be statically linked to the other two subprojects.

2.1. Package Dependency

\[\text{Figure 1. Package dependency}\]

---

\(^1\) Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley. ISBN 0-201-63361-2.
3. Mastergoal Subproject

3.1. Board Status Package

3.1.1. Structure

Class Piece: This is an abstract class that represents the top of the pieces hierarchy. It represents an object that has a location (position) and can be located in the board. The position is an integer that represents an index in an enumeration of positions of the board. This is explained later in the class TPosition.

Class Ball: This class represents the ball. It extends the Piece class.

Class Player: This class represents a generic player (token) of the game. A player has a team.

Class Goalkeeper: This class represents the goalkeeper token in the board. It specializes the Player class since the goalkeeper has properties and moves that are different than the other players of the same team.

Class Team: This class represents a team in a Mastergoal game. A group of players belong to a team.
Class Play: This class represents a sequence of moves that are done by a team in a turn. The play can be a simple move of a player, a kick of the ball or a series of passes of the ball terminating in a final kick. A play consists of one, two, three or four moves. Figure 4 shows an example of a play.

Class Move: A move represents a single move of a player or the ball. A move has a piece (ball or player) and the corresponding starting and end position of the moved piece.

Class Game: The game class represents a game in the Mastergoal environment. The game class is a controller class that serves as a link between the components of a Mastergoal game. The game class manages the turns between the teams, maintains the current board status and the list of plays done by the teams.

Class Board: The board class represents the status of the board in a Mastergoal game. It contains operations to compute the different areas of the board and positions of the pieces.
Class ABitBoard: This class represents a board as a bit array. The actual values are held in the value member (an array of 32 bit integers). This array maintains a bit for each position of the board plus a column and row on each border of the board to represent the out of bounds position, as can be seen in Figure 7. The array size must be then of \((11 + 2) \times (13 + 2) = 195\) bits (11 columns, 13 rows plus the surrounding border). Since the array is of 32 bits integers, we need 7 integers to hold all the positions.
Class TPositions: This class is a class with only static operations and static data. This class represents each of the positions of the board. It does so by holding a (static) array of ABitBoard (boards) see Figure 6; this array has 195 components, one for each position. Each member contains a board with only one bit set. This bit represents the bit of the position in the board: i.e. The first component will contain a board with only the bit representing position 0 set. The second component will contain a board with only the bit representing position 1 set, and so on. This sacrifices some memory but is handful to query information about board status by doing bit operations between the boards.

Class MGRules: This abstract class represents the root of the rules hierarchy. The game class and the MGRules hierarchy implement the “strategy” design pattern. The Game class constructor receives an appropriate object of the MGRules hierarchy which will be used as the rules for the game.
**Class AbstractMtaskgoalFactory:** This class represents the root of the Mastergoal Factory hierarchy. Classes in this hierarchy are responsible for creating rules for each level of the game. E.g. MastergoalFactoryL1, MastergoalFactoryL2, MastergoalFactoryL3.

**Class MGRulesL1:** Rules for level one.

**Class MGRulesL2:** Rules for level two.

**Class MGRulesL3:** Rules for level three.

![UML Diagram]

**Figure 9. Turn management. Implements the “Observer” design pattern**

The TurnSubject, Game, TurnListener and AbstractAgent classes implement the “observer” design pattern.

**Class TurnSubject:** This abstract class implements the Subject role in the observer design pattern. It notifies of turn events to the listeners. The Game class implements this role extending this class.

**Class TurnListener:** This abstract class implements the role of the Observer role in the observer design pattern. It expects turn events from the TurnSubject.

**Class TurnEvent:** This class represents an object that contains the turn data passed in the turn event.

**Class AbstractAgent:** The abstract agent is an abstract class that represents the root of the Agent hierarchy. An agent is an entity that plays the game in behalf of a team. It can be a computer program, a human program, a network connection, etc.
**Class HumanAgent:** This class represents a human agent.

**Class MachineAgent:** This class represents a machine program playing the game.

---

![Diagram](image)

**Figure 10.** Agent factories.

**Class AbstractAgentFactory:** This abstract class implements the abstract factory design pattern for creation of agents. Specific subclasses create specific agent objects.

**Class HumanAgentFactory:** This factory class creates human agents.

**Class MachineAgentFactory:** This factory class creates machine agents.

**Example Object diagram:**

![Diagram](image)

**Figure 11.** Game example object diagram for game in level 2.
Figure 11 shows an example configuration for a game in level 2. In level two the teams are limited to two players each and there is no goalkeeper.

### 3.1.2. Behavior

#### Create game diagram

In this section sequence diagrams are presented showing the interaction between the objects in some representative use cases.

![Game creation diagram](image)

The previous diagram shows the objects and messages involved in the creation of a game.

#### Play game diagram

Figure 13 shows the interaction between objects involved during a game.
Figure 13. Start game use case.
4. MGAI Subproject

4.1. Structure

Figure 15 shows an example of the relation between objects for the search configuration of an agent. This example uses a MTDF as a search algorithm and a linear strategy.

The classes AbstractAgent, SearchStrategy, MTDFSearch and AlphaBetaSearch implement the “Strategy” design pattern, with the Search hierarchy implementing the strategy and the AbstractAgent implementing the context.

**Class Node:** This class represents a node in the search tree. A node contains a pointer to a board, has a depth in the search tree, a value and a reference to it’s children and ancestor (parent).

**Class SearchStrategy:** This abstract class is the root of the search class hierarchy. All subclasses implement a search algorithm. The search must run in a separate threat to allow the program to run while the search is being done.

**Class MTDFSearch:** Implements the concrete MTDF search algorithm.

**Class AlphaBetaSearch:** Implements the concrete AlphaBeta search algorithm.

**Class MGAbstractStrategy:** This abstract class implements the concept of an evaluation function. Concrete subclasses implement the Compute method in different ways.

![Figure 14. Search context](image-url)
**Class MGLinearStrategy:** This class represents the concept of a linear evaluation function. It contains an array of weights, one weight for each term.

**Class MGNNStrategy:** This class represents the concept of an evaluation function based on a Neural Network algorithm. It is given in this document as an example: No further design in this area is presented.

**Class Term:** This class represents a particular feature of a board. A term object has a weight that is the associated weight of the term in a strategy.

![Diagram](image.png)

**Figure 15. Search example.**

### 4.2. Behavior

Creation of a Machine Agent.
Figure 16. Association of search context with a Machine Agent
5. MMLE Subproject

5.1. Model Package

5.1.1. Structure

Class Framework: The framework class is the controller class for the machine learning environment. The Framework creates experiments using the “Abstract Factory” design pattern.

Class AbstractExperimentFactory: The abstract experiment factory implements the Abstract Factory role in the “Abstract Factory” design pattern.
Class GAExperimentFactory: This class is a concrete Factory class that creates GA experiments.

Class AbstractExperiment: This class represents the root class of the experiment hierarchy. The framework can have many experiment types, all of which should specialize this class.

Class GAExperiment: This class represents a specialization of the AbstractExperiment class for GA experiments.

Class TerminationCriteriaManager: This class represents an object that load dynamically linked termination criteria.

Class FitnessFunctionManager: This class represents an object that load dynamically linked fitness functions.
5.2. **GA Experiment**

![Diagram of GA Experiment classes]

**Class Population:** This class represents the population (set of individuals) of a particular experiment at a given time.

**Class GAData:** This class is a model class that contains the data (population number, crossover rate, etc) of a GA.

**Class Individual:** This class represents an individual in the genetic algorithm context. An individual is associated with a strategy and holds the current fitness of that strategy.

**Class AbstractTerminationCriterion:** This is an abstract class that represents the root of the termination criterion hierarchy. Each subclass should implement a concrete termination criterion algorithm.

**Class AbstractFitnessFunction:** This is an abstract class that represents the root of the fitness function hierarchy. Each particular subclass should implement a concrete fitness function algorithm. See Figure 18
**Class SelectionContext:** This is an abstract class that represents the interface for classes that select individuals from a population. Together with the SelectionCriterion hierarchy implements the “strategy design pattern”. Implements the method `SelectIndividuals()` which selects n individuals from the current population.

**Class SelectionCriterion:** This abstract class is the root of the selection criteria hierarchy. Each individual class that inherits from this class should implement a concrete selection criterion algorithm.

**Class RankSelection:** This class implements the concrete rank selection algorithm.

**Class FitnessProportionateSelection:** This class implements the concrete fitness proportionate selection algorithm.

**Class TournamentSelection:** This class implements the concrete tournament selection algorithm.

### 5.2.1. Behavior

In this section a number of sequence diagrams that show the interaction between the objects of the previous classes are presented:
Train Strategy:

Compute Fitness:
5.3. **User Interface**

The user interface is coded using the Microsoft Windows Forms framework. Figure 24 shows the dependency among the classes. A prototype of the user interface is included as a separate artifact.
Class FrmMain: This form represents the initial form that is displayed on the screen and allows the selection between the different actions that can be done in the framework.

Class FrmGame: This form represents the window in which a user can play a game against another user or a machine player.

Class FrmExperiment: This is the form in which experiments are created. Only general data of experiments is used in this form. Specific data of specific experiments is loaded/viewed in a specific form. See FrmGAPopulation.

Class FrmGAPopulation: In this form the information specific to GA experiments is loaded/viewed.

Class ExportStrategy: This form is used to export strategies to external files.