BW Cooperative State University Mannheim, Germany

Knowledge Based Systems Report

Learning play finder for complex models and complex offensive play for SSL RoboCup

Based on and integrated in central software of the team Tigers Mannheim

Authors:	**Ulrich Ahrendt, Sebastian Nickel, Felix Pistorius, Fritz Reichwald, Nicolai Ommer, Dirk Klostermann, Daniel Andres Lopez**
Course:	TAI10ABC
Study course:	applied computer science
Study manager:	Prof. Dr. H. Hofmann
Tutor:	Paul Hubert Vossen
Semester:	5. Semester
Date:	14/11/2012

Document revisions

Rev.	Date	Description	Editor
1	11/9/2012	initial, short project descriptions	Nicolai Ommer, Dirk Klostermann, Sebastian Nickel, Felix Pistorius, Fritz Reichwald, Ulrich Ahrendt, Daniel Andres Lopez
2	12/10/2012	general concepts for play finding agent, basics	Nicolai Ommer, Dirk Klostermann, Fritz Reichwald, Daniel Andres Lopez
3	21/10/2012	first approach of play described, basics	Sebastian Nickel, Felix Pistorius, Ulrich Ahrendt
4	05/11/2012	restructure to add details of the projects and the implementation	Dirk Klostermann
5	10/11/2012	details of implementations and used algorithms, basics	Nicolai Ommer, Dirk Klostermann, Sebastian Nickel, Felix Pistorius, Fritz Reichwald, Ulrich Ahrendt, Daniel Andres Lopez
6	12/11/2012	complete and finalize chapters, added additional diagrams and graphics	Nicolai Ommer, Dirk Klostermann, Sebastian Nickel, Felix Pistorius, Fritz Reichwald, Ulrich Ahrendt, Daniel Andres Lopez
7	14/11/2012	Latex setup, import of data in Latex document, final arrangement	Daniel Andres Lopez

Contents

List of Figures

List of Tables

Part I.
Basics

1. Introduction

The RoboCup is an event for robots playing soccer against each other. There are different leagues with different kinds of robots. This project will deal with the Small Size League (SSL). There are 6 small robots in cylinder form per team. They are controlled by a central computer per team. The field is detected by two cameras and a special software will process the images and send the data via network to the team computers, so that teams do not have to deal with image processing. Additionally there is a referee software that is currently operated by a human referee who sends commands to the teams. It is not allowed to interact with the team computer while a game is running (only during a timeout/break), so the software must be completely autonomous. The software uses a play/role pattern. A play defines roles and how these interact with each other and a role will be directly mapped to a bot.

The team Tigers Mannheim participated at the World Cup in Istanbul and Mexico and is looking forward to participate in the Netherlands, too.

2. Sumatra - central software

Sumatra is the central software of the Tigers. It is responsible for receiving vision data from SSL vision and referee messages from the referee software as well as sending commands to the bots. The bots themselves are not intelligent. Figure 1 shows the general structure of Sumatra and its interfaces to the outside. This project will be completely located in the AI module. More aspects of the general concepts of Sumatra will be discussed in [1].

3. Basics

3.1. Set of plays

The following list shows some possible plays to choose from. They can be divided into different groups:

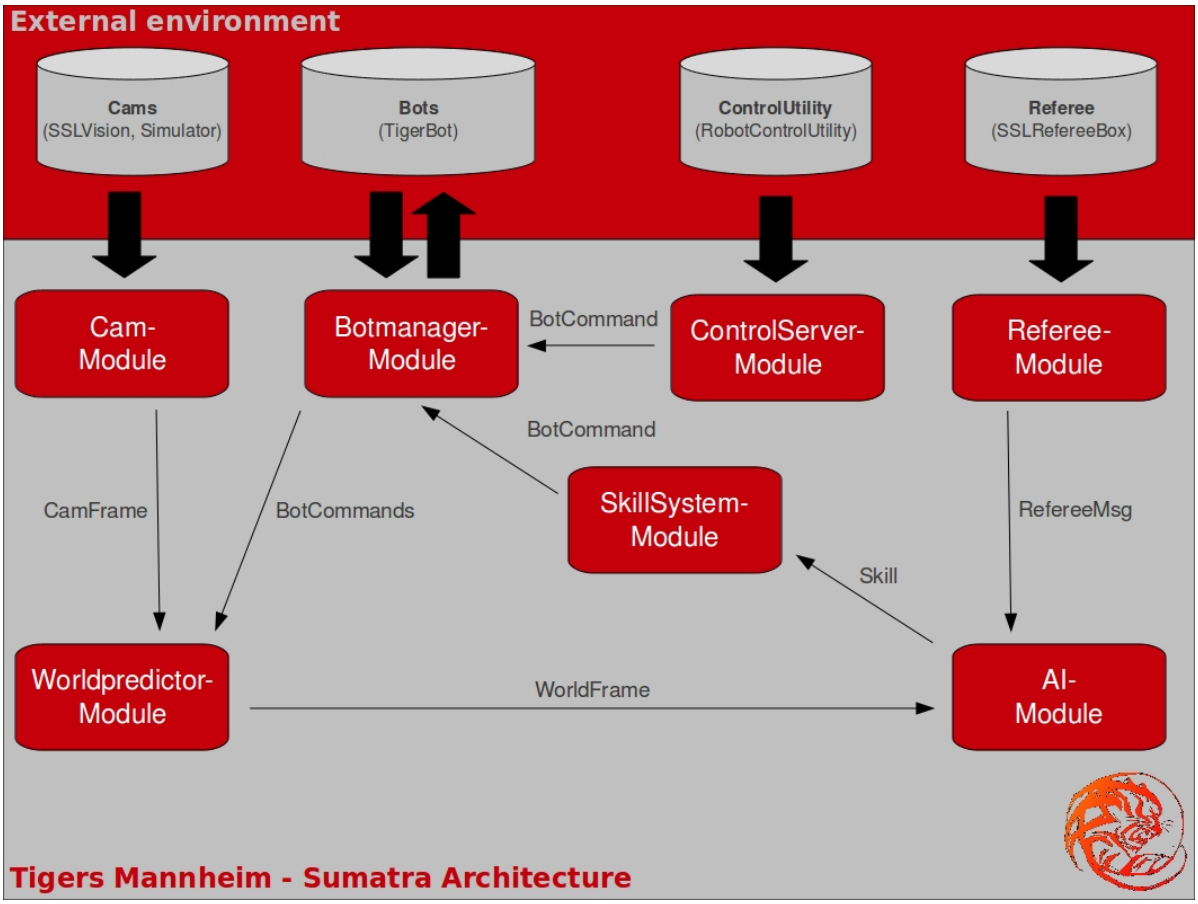

Figure 1: Architecture Overview of Sumatra

3.1.1. Offensive plays

- hidden shot
- direct shot
- indirect shot
- get the ball
- passing

3.1.2. Support plays

- break clear
- Man to man marker

3.1.3. Defensive plays

- Keeper with one defender
- Keeper with two defenders

4. Play finder

While all Bots are controlled with one Software, Sumatra, it has to determine (see figure 2 for overview of information flow), which bot gets what specific task. Because bots have to play together, the tasks are combined to plays (this concept is explained in chapter 11 Play-Role Pattern). A play finder is responsible for choosing a valid play from all plays suitable for the current situation. There it has to consider different circumstances, which are discussed in chapter 6 PEAS Analysis. At the moment there is already one simple play finder implementation. It is a static set of rules, which defines for each situation a specific set of plays. This rules are directly implemented into the code and for each situation, the developer has to make sure, there are enough plays so that each bot has a role. Too many or too few roles will cause problems, because plays cannot operate correctly or bots are standing around without a task.

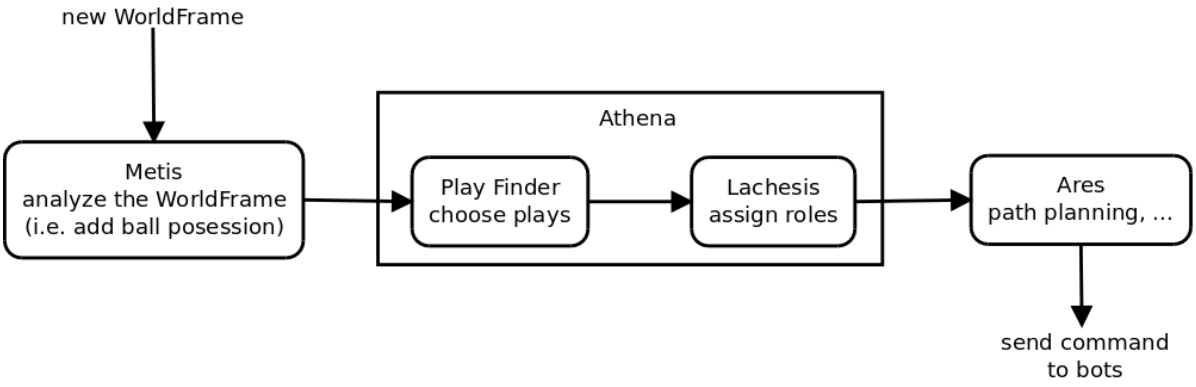

Figure 2: Information flow in Sumatra

The play finder implementation is easily exchangeable. A play finder implementation has to implement a method for choosing plays in a normal situation without a special referee command. Additionally, there are lots of referee commands that require special plays. To avoid redundant code, a play finder may decide to implement reaction for those signals or not. A default implementation is already given by the underlying layer.

This is, because at the moment there are not many alternative plays to decide between. So here, we plain old static role approach will stay.

A point worth considering is, that the play finder also has to react on removed and added bots as well as the current bot number in general. This means, the set of plays has to be different for different bot numbers.

Part II.
Project Play Finder

The first project is to develop a new play finder. It should be more intelligent as the existing one and not static anymore.

5. Current situation

There is a basic play finder, that decides, which plays are chosen in a match. It actually consists only of two methods:

- choosePlaysFreely
- reactOnRefereeCmd

Both methods are processed linearly. Besides, for each number of bots, there is a separate branch. This hard coded Play Finder is able to choose to correct plays for referee commands. During the match, it simply checks if there is a direct line to the goal and chooses a direct or indirect shot accordingly. It is in the developers responsibility to set the right amount of plays, so that each bot gets a role. If a play gets an additional role, the Basic Play Finder has to be adapted.

6. PEAS Analysis

The PEAS analysis gives a full description of the important parts, which the new agent has to consider.

Performance measure

- Fail of offensive or defensive play in current play selection
- Success of offensive play
- goals
- referee interception (yellow cards, etc.)

Environment

- field with 4x6m, green floor
- lines that mark the halves, the kickoff area, the defense area, the penalty mark and the field borders.
- ball
- foe bots
- Light

The environment is fully observable with all sensors. It is not deterministic, because the influence of the floor is different for several fields and therefore the movement of the bots and the ball is different. Also it is hard to determine the play strategy of the foe bots. Due to the fact that the opponent team could also be seen as an agent, the environment is not single agent based. Since Sumatra only reacts on discrete information of the Vision it is also a discrete system. It has definite states.

Actuators

- Bots
  - 4 Wheels with an own device
  - Kicker
  - Chip kicker (not yet)
  - Dribbling machine

Sensors

- Vision
  - 2 Cameras above the field, which record the field and preprocess the data, to assign each bot an ID (by a colored pattern) and to detect the ball.
- World Predictor
  - calculates a future, stable state based on the current state given from the vision, so that the AI System could do its calculation

- Bot
  - kicker charge
  - battery status
- Messages from the referee
  - Commands (Start, Stop, Kickoff Enemies, Penalty Tigers, ...)
  - Information (Time, Goals)

7. Learning agent

The new play finding algorithm will be designed as a learning agent. That is the opposite of the old play finder "BasicPlayFinder", which encode reactions for common situations. Since a game consists of two teams, each with 6 bots, and one ball, there are many possible situations that have to be evaluated. This can't be done in a general or encoded way. A play finder should also be extendable for more input variables, like the velocity of the ball or a bot and the orientation in the field, and for own bots internal data like state of the kicker module and so on. According to all these variables a situation is always a new situation, that has not be occurred before (it is assumed that the really minimal probability of the same situation, which means all variables are equal, don't occur). This leads to the point that the agent always has to handle an unknown situation, therefore it only can use probabilistic behaviour, which means that the agent learns and compares current situation against its experience. The agent should have to possibility to save its knowledge, because there are too many situations, that modifying the agent itself would not lead to a more generalized agent. With a comparing approach of the experience a wider range of new situations could be handled, because it is possible to integrate dynamically learned rules for accepting the comparison. Figure 3 shows the concept of a learning agent. It is adapted this agent and will be described in the following chapters.

7.1. Design of a learning component

The play finder will be designed as a learning play finder to find the next plays to execute. Not all situations can be used for a learning based agent. Some require a special treatment. Because on some referee signals the plays don't have to be chosen, they are already fixed. Therefore the learning part will focus on in game situations.

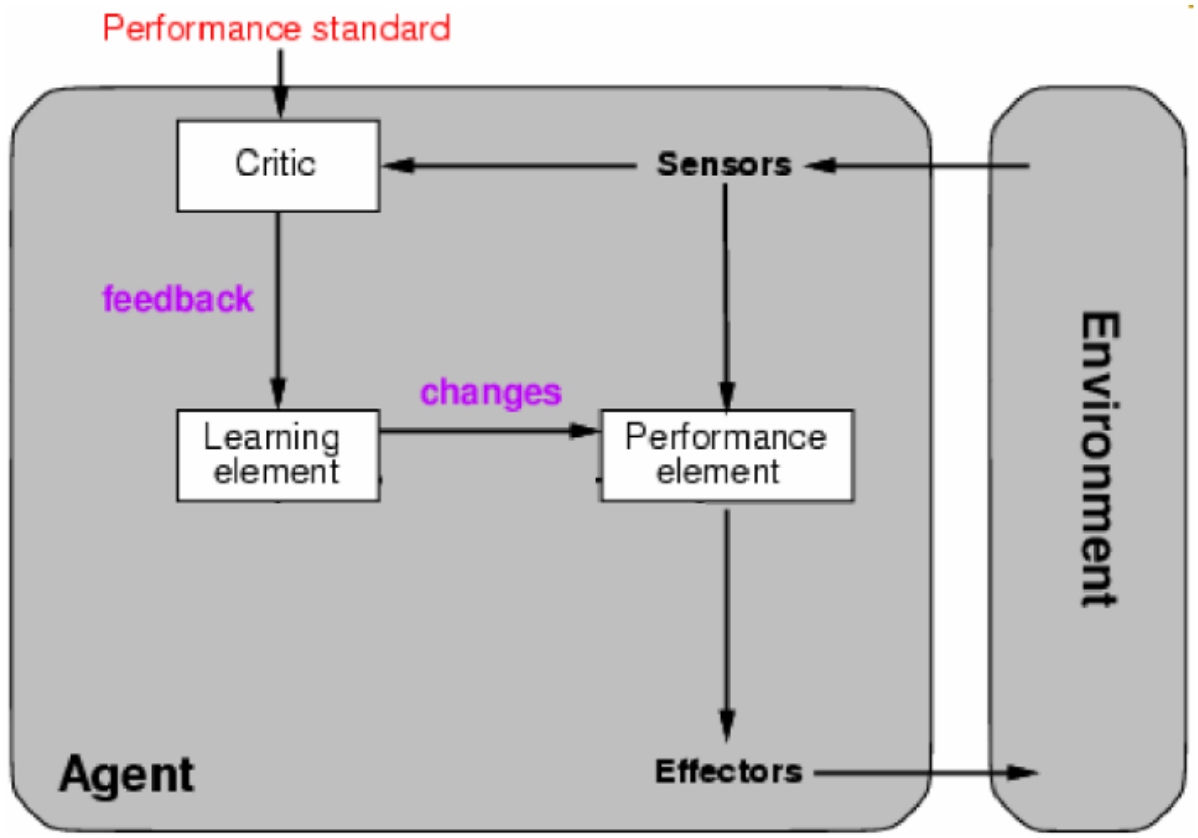

Figure 3: General concept of a learning agent
Source: [2, p. 6]

Each situation consists of a several attributes. The agent will have access to the information that was recognized by the cams and prepared by the world predictor (an internal component that prepares the raw vision data for further processing). This information consists of the positions and velocities of all bots, owns and foes, also the position and velocity of the ball. Additional information about the own bots is available, like the charge of the kicker module. All current plays are also available. The attributes that will lead to the learning are the states of the current plays, especially of the offensive play. The other performance measures mentioned in the PEAS analysis could not used as so effective as the state. This causes in the delay that the information is available. The state of a play is determined by the play itself, it has its own performance measures and will end with failed or succeeded. The referee signals and the goal counter are send by an human referee. Therefore there is a delay and the play finder, could already chose other plays, so these signals can't be used to learn of a situation.

7.2. Type of learning feedback

The learning agent will use supervised learning to build up its knowledge. This concept will help the agent to sort its decisions. Since the situations have to be compared against the current one and the best matching is chosen, the agent has to know whether this selected situation and its results was successful or not. Therefore it will save the current situation with the selected plays (the decision) and the results of the play together.

7.3. Matching algorithm

The two main concepts for learning agents to match their current observation with their knowledge are Naive Bayes Classifier and Decision Trees. But both don't fit to the current problem. Naive Bayes Classifier use probabilities to determine whether the acceptance or the discarding of a possible decision should be made. This approach could also combine several input attributes. A main problem of this technique in the context of play finding is, that it isn't possible to determine possibilities of attributes without calculating the future, because a current observation is not the same as before and therefore the knowledge can't be easily and performant used to find a possibility of a decision. The problem with decision trees are nearly the same, but there is also another one. The tree would not be binary. For searching a decision this isn't a problem, but for building it. The agent has to determine, where to append the tree and sometimes what to combine or separate. This is not a trivial part, because of huge possible amount of situations.

There are two approaches that fit for a matching algorithm for a learning play finding agent. The first one is two order the old situations by its similarity to the current field, this needs for each situation a new calculation in a decision. The other used the nearest neighbor method, therefor it is necessary to find unmodifiable values over time for a situation, that don't depend on another situation. Both concepts are explained in detail in chapter 8.5 Comparing field situations.

8. Learning Play Finder algorithm

8.1. Idea overview

The Play Finder gets as input the current field state and all history states of the field in combination with the chosen play in that situation and whether it was successful or

failed. Based on this input data it will determine probabilities for sets of all available plays and then choose the set of plays with the highest score. Furthermore, it can be regarded if the score increased in the near past, which is an indication that the actual situation is not yet reached but it will be reached in the near future. The play is then executed and reports its result back to the Play Finder, so that it can be saved with the field information in the history. Figure 4 visualizes this.

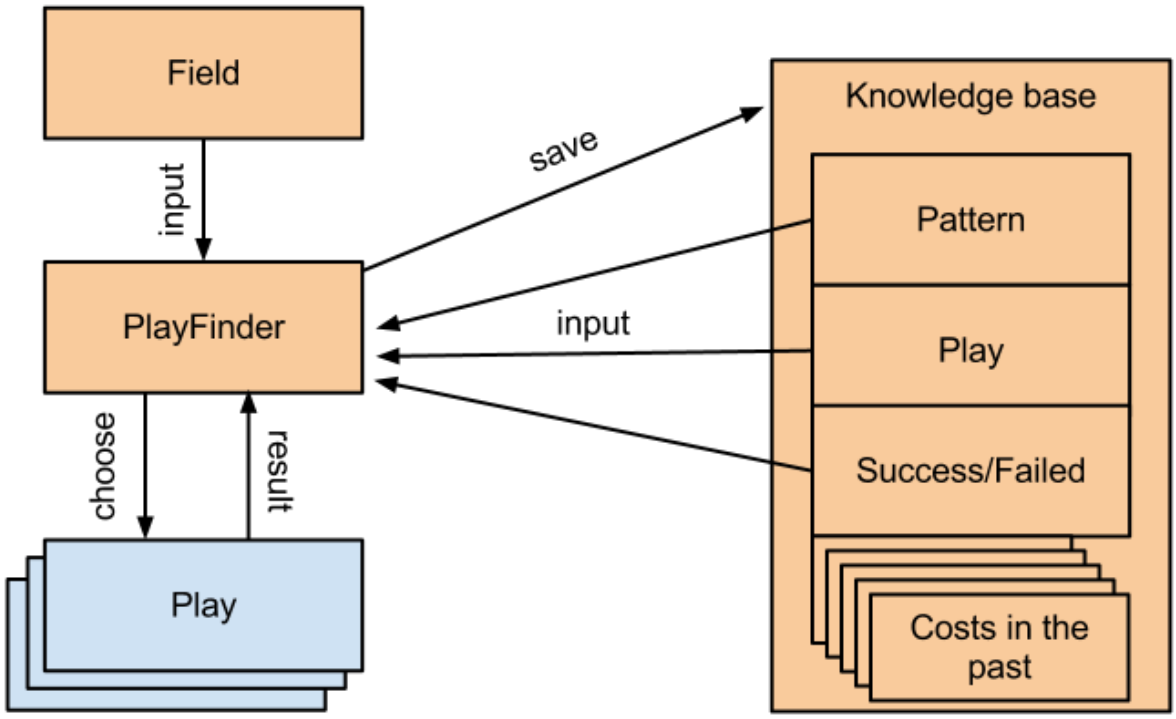

Figure 4: Overview of the learning Play Finder

8.2. Requirements and considerations

There are some points that should be considered during this project: The PlayFinder component consists of four classes. The first one is an abstract implementation the APlayFinder. The abstract class covers all actions on referee commands and the force new decision with default implementations. Further implementations are done in the BasicPlayFinder that inherits from the APlayFinder and is the old not very intelligent implementation. Now there is also the LearningPlayFinder that is an intelligent implementation that chooses the plays not only from actual input but with information from earlier actions that are stored in the knowledge base. The fourth class is the PlayFind-

erAdapter. This class decides what the LearningPlayFinder has to do in context with the information that is provided by Sumatra

- avoid hard coded decisions: find a way to map situation to plays in config file
- keep it simple, so everyone can easily insert new plays
- always fill all available roles
- security: some referee commands have high priority, e.g. HALT
- maybe consider score

At the moment the Learning Play finder chooses at one situation for example 3 Plays. One contains 3 roles and is an offensive attack of the opponents part of the field. One of the remaining bots gets the keeper play and the remaining 2 will protect the own part of the field so at this moment there are 3 active plays. If the offensive play fails, it is marked as failed play, the data is stored in the knowledge base and the PlayFinderAdapter starts the process of play finding again with the new provided information.

8.3. Building up the knowledge base

The foundation of the decision for a play is the knowledge base. This is the brain of our learning play finder. It stores an amount of field situations (position and angle of bots, ball position, ...) which occurred in the past. This situations are assigned to a play, which was chosen in this certain situation. Furthermore it is stored, if the play was successful or not.

The knowledge base is filled during a game. Whenever a play is chosen and executed the result is saved in the knowledge base afterwards. Of course, at the beginning the knowledge base is empty, so it cannot use it for choosing a play. To have a starting point the plays are chosen randomly first. If no situation from the knowledge base matches to the current state the plays are chosen randomly. On this way, the play finder keeps on learning. If there is an unknown situation the play finder will try something and afterwards it is more intelligent because it knows the outcome of the combination from this situation and the chosen plays.

On termination of the application the knowledge base will be persisted to a file. At the moment this is done by serialization to XML files. When the play finder is started again, it will load the saved file and can continue with the previous knowledge base. This mechanism is necessary, because the agent will not always run.

8.4. Process to choose the plays

The implemented play choosing algorithm is triggered for every new world frame. First, it has to be checked if new plays have to be chosen. Otherwise, the play finder does nothing. Reasons for the force of a new decision are:

- A play succeeded or failed (The play itself returns this)
- A referee command enforces new plays (i.e. a free kick, goal,...)
- The play finder itself decides that it is time for new plays (timer)
- The user enforces new plays (not allowed in a game)

When a new decision is forced the "offensive" play is chosen first. This is the play which cares for the ball. If a bot of the tigers has the ball this will be a shooting play or a play which tries to improve the positions of the bots and the ball for shooting a goal. If the Tigers are not in ball possession a play is chosen to get the ball. Next, the "defensive" play is chosen. This play has the task to protect the goal. It contains the role for the keeper. Last, the other bots get support plays. This is done in a loop by assigning support plays as long as there are bots without a role. Figure 5 shows the described method in an overview.

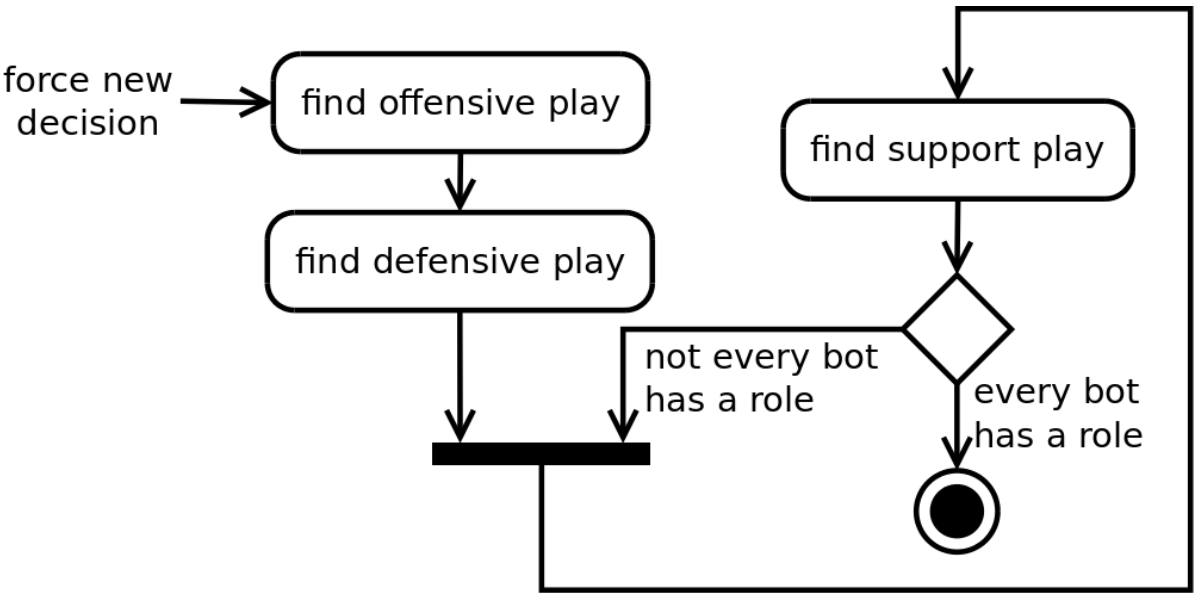

Figure 5: Order of play types

The methods to find a offensive, defensive and support play are the same. The only difference is the set of the considered plays. This method does a preselection of the plays first. This means that every play is asked if it can be used in the current situation. For example, a shooting play would deny this if the enemy has the ball.

Next, the current field stored in the world frame is compared with the successful situations of the preselected plays. These situations are saved in the knowledge base (compare figure below). It is searched for the field with the highest similarity to the current field. This indicates that the according play worked quite well in a situation like this. Next, it is checked if the current situation has a high match in a failed situation of this play, too. This has two reasons:

- Perhaps exactly this situation was already chosen once. For example, if the current field was matched to 90% with a successful situation of a play it sounds like a good play to choose. However, if there is a 99% match for the same play but in this situation the play failed it does not sound so well anymore. In this case we already tried this play in nearly exactly this situation and it failed.
- It is not sufficient to rely only on the successful executions of the plays. The major part of the plays fail and there are not so many games. So there will not be a huge amount of successful executions for each play. It is necessary to get information from the failed executions, too.

If there was no high match among the failed situations the play can be chosen. Figure 6 shows an overview of the described algorithm. A more detailed activity diagram is attached in the the appendix A as figure 16.

8.5. Comparing field situations

A field situation is a combination of bots from two teams and the ball. In order to choose plays according to a similar field situation, those must be comparable. This chapter will outline some approaches and ideas that were considered, tested and finally implemented.

The overall problem when comparing those fields is, that it must be very efficient as lots of fields have to be compared. Furthermore, field situations should be as unique as possible to avoid accidental mismatches. Also it is not enough to say if two field situations are equal or not, but factor of how equal they are is required, to also choose field situations that are similar, but not equal, to the current situation.

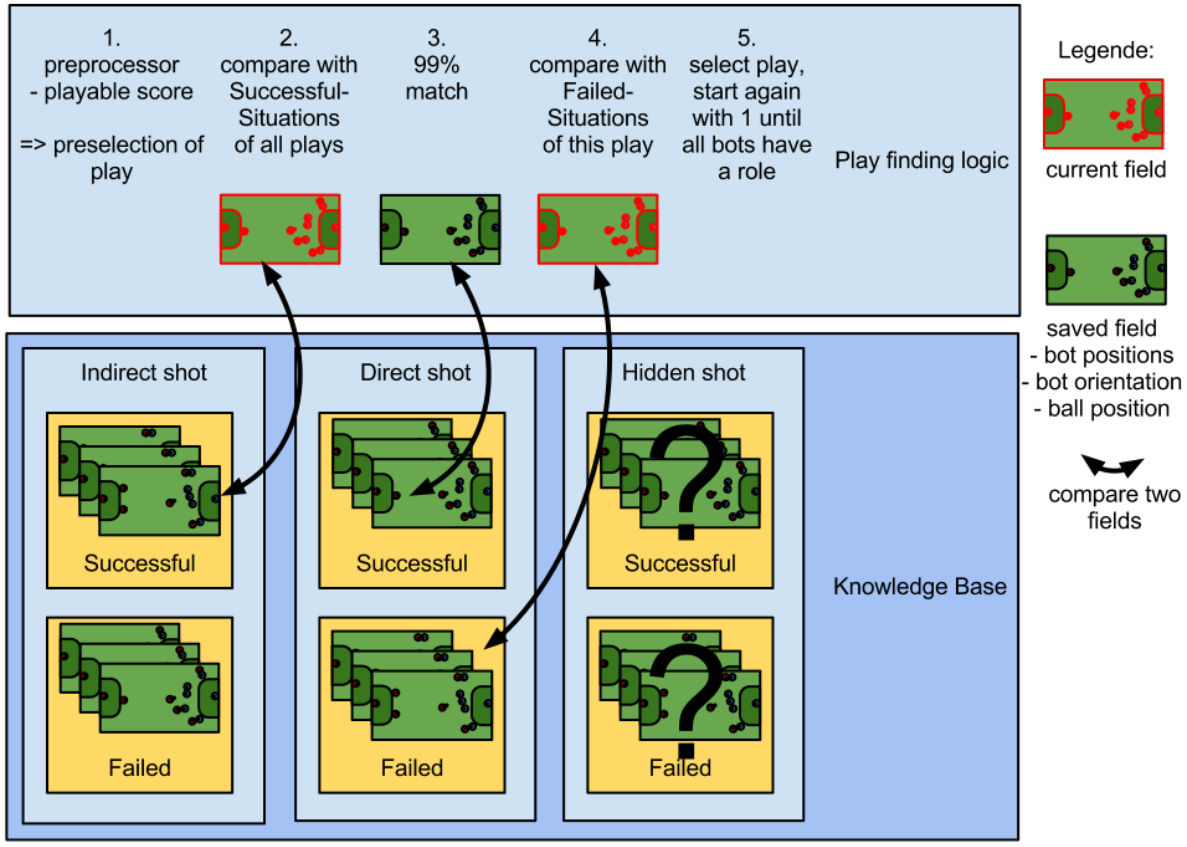

Figure 6: Overview of the play finding algorithm

With the **Nearest Neighbor Method** it is possible to check the new data against the knowledge base (an example is for two dimensions is shown in figure 7). But this approach will also works if there are static results available, that don't change in new situations. Therefore this method could only be adopted for comparison algorithms that uses scalars or vectors that identify a field. It is not possible to work with this method for a situation based comparison, i.e. for comparing each field with the current one. If a comparison algorithm will result in a n-dimensional vector, rules for defining the nearest points have to be made. This will need more investigation and will be done if there is such a practicable algorithm.

8.5.1. Idea 1: Chain the bots together from left to right and calculate distance

The first step when comparing a field situation is to map each bot from the new situation to a bot from the old situation. Especially when the two situations are not that similar,

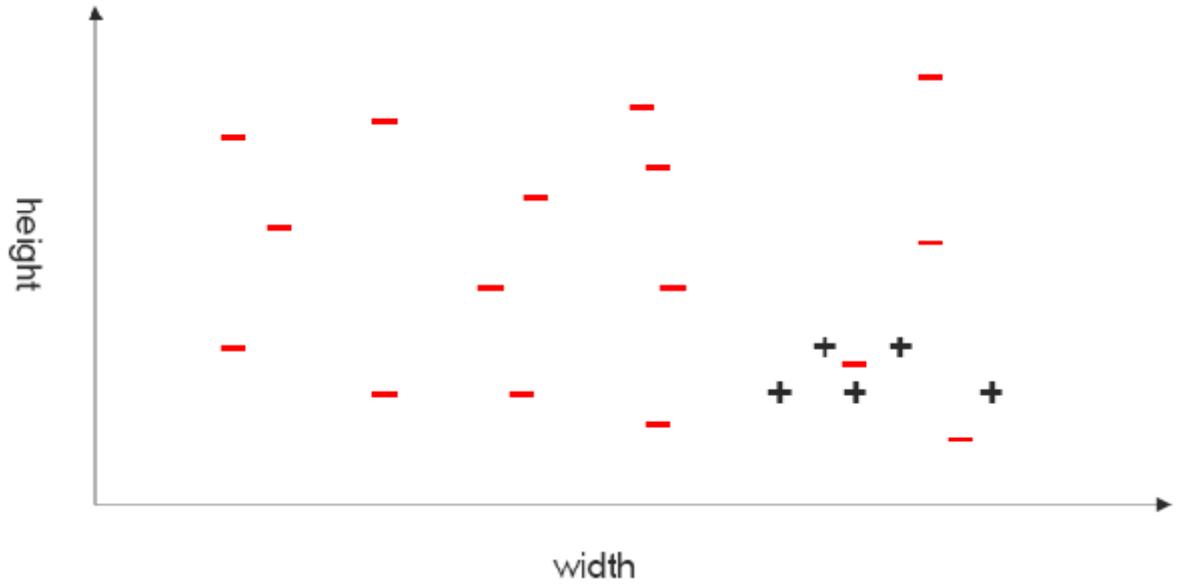

Figure 7: Nearest Neighbour method example with two dimensional vector
Source: [2, p. 13]

this is not an easy task to do, because for a computer it is hard to decide which bot will belong to which without overlapping. The upper part of figure 8 illustrates one possible solution of how bots may be mapped. Starting from the left, they will be chained together from leftmost to rightmost. This will be done for the new and the old field situation. This will give us two ordered lists. Then, the distances from old to new position of each bot will be calculated in the same bot order. This is may not be the optimal mapping, as for example, the mapped bots may be on opposite sites (on top and on bottom in the picture).

Having the distances between the bots, we now need a way to evaluate the similarity of those values. The most efficient and simple way would be, to just sum up the distances. However, this would not lead to a unique comparable factor. A single number can hardly represent a complete field situation as long as it does not have a state for each possible situation. Also, the similarity will not be very representative as well.

A better solution is, to compare the distances between the bots with the maximum possible distance, which is the diagonal of the playing field times the number of bots. This will give a percentage of how similar one field situation is to another situation.

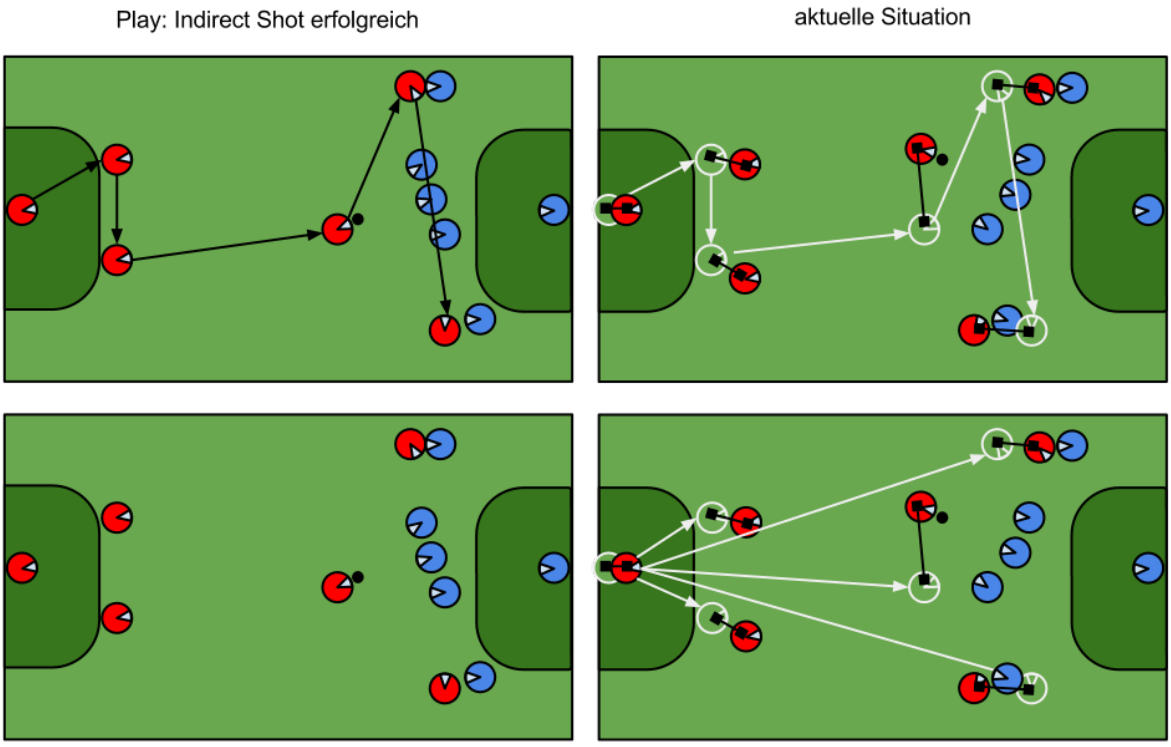

Figure 8: Comparison of fields by distance difference between bots

8.5.2. Idea 2: Extend first idea by calculating all possible new bot ¡-¿ old bot combination

The second idea, that is based on the first one, is to try all possible bot combinations from old and new field situation and find the smallest sum of all distances. It is illustrated in the lower part of figure 8. All possible mappings from the bots in the one situation to the bots in the other situation are build and for each mapping, the costs, namely the distances are calculated. The distances will be summed up. At the end, the mapping with the lowest sum will be selected, as in this mapping, the bots have in average the shortest path to their mapped bot. This is exactly the way, the role assigner will choose the right bot for each role. Just that there are roles with initial destinations instead of a second field situation. This approach is very time consuming. It is OK for the role assigner, but comparing fields will be done quite often in each frame. So here is a great performance issue, but the mapping is the most unique from all methods mentioned in this report. Tests with this implementation showed, that it will take about 1.5s for 10,000 fields. Compared to the available 16ms per frame, this is far too long, even when the number of fields is dramatically decreased.

8.5.3. Idea 3: Use geometry

The geometry idea is an approach that each field could be uniquely described by a small set of values, that are calculated once. Based on the approach that a triangle has an incircle (circle that touch each vertice once), an outer circle (a circle with the circumcenter as middle point) and the nine-point circle (a special circle with a combination of centroid, orthocenter and circumcenter and eulers line), the idea is to find such middle point and radiants also for a polygon, which is defined by the position of the bots. The middle points and the radiants are saved ordered for each field. For a new field these values are calculated. With the nearest neighbour method the best matching field can be found. If the result of the selected plays is known, the field will be inserted at the correct place in the ordered set. This approach will be an efficient algorithm, because a field is classified by its geometric values and new input only has to be matched by the nearest neighbour method.

The problem here is that these values are not proved to be unique and therefore it could lead to wrong decisions, although the set of 3 circles and 3 radiants is quite a lot. But the 6 values have to be ordered in a 6 dimensional body, so that the nearest neighbour method could be applied. Another problem is that the incircle, outer circle and nine-point-circle are only well defined on a triangle. For polygons there are no definitions for all circumstances. In most cases they have to be convex, but this cannot be guaranteed. Also, different numbers of bots will also produce other geometric scalars. Therefore for each bot number an own vector room has to be saved. Additional, for each field the algorithm has to save also the ball properties for a classic compare, since this is not included in the geometric scalars. Extending this algorithm with more parameters like velocity and the angle of the bots could not easily be done, because they need to have a geometric meaning. At the moment a bot is represented as a dot in this model and the model has to be extended to more dimensions.

Due to the many known problems, especially the missing expandability and missing well defined formulas for all circumstances, this approach will not be implemented. Because of this decision a description of the mathematical concepts will be omitted.

8.5.4. Idea 4: Use the available field raster

Sumatra already contains a module that is able to analyse a field raster that represents occupation of teams on the complete field[3]. It will span a dynamic raster over the field and allows a value of 0-100 for each cell, where 0 represents full occupation of the one

team and 100 full occupation of the opponent team. A value of 50 indicates equal or no occupation. A graphical representation of the field analysis is shown in figure 9.

Looping over the raster and comparing differences between two field situations is quite fast and can be optimized by modifying the size of the raster, so that performance should not be a great issue with this implementation. Furthermore, the raster will already be calculated in each frame, so that the information are available without any further computation.

The way of comparing to analysing field is, to compare cell by cell. This means, we start with the upper left cell on both field situations and calculate the difference. The differences for all cells will be summed up. The similarity factor is received by the sum divided by hundred times the number of cells, namely the maximum possible difference.

The occupation on the field is one of the most important indicators for a similar field situation, so the raster will deliver quite a good data source. However, special play patterns may not be detected. Also, there is the small chance of two rather different situations are found to be similar, as matches might be ambiguous. This is, because it will not make a great difference, on which absolute positions the bots will be located on the field as at the end, all differences will be summed up.

As this approach is a good starting point, it was chosen for final implementation with the idea of extendability of the raster in later releases.

8.5.5. Further considerations for the future

The comparison of field situations is a fundamental part of the overall project. A lot of optimization and alternative Implementations can be done. So from the beginning, we decided to build everything very modular. It is possible to place alternative Implementations into the code without modifying any existing code. For now, however, there is a working and efficient implementation that will last for the first version of this learning play finder.

8.6. Performance issues

The idea to compare all situations in the knowledge base with the current field is not realistic. Unit tests revealed that it dues up to milliseconds to compare only some hundreds of situations. However, the upper limit for the whole Artificial Intelligence is 16ms for each frame.

There are two possibilities to reduce the amount of field comparisons. First, all the

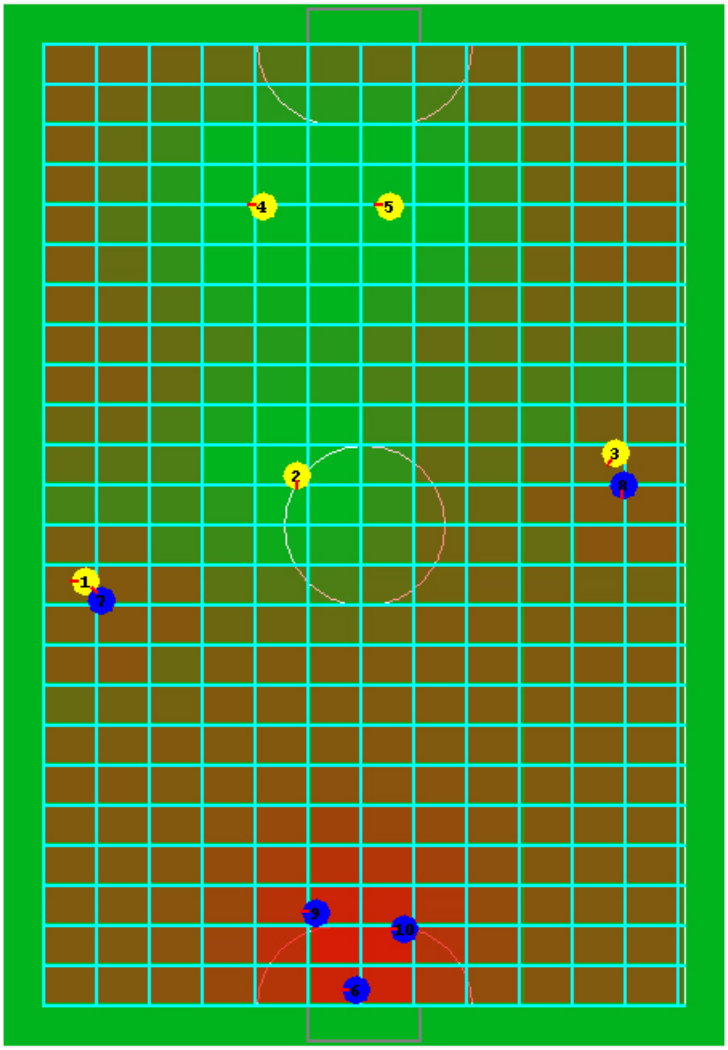

Figure 9: Example for a rastered field

situations according to a play could be sorted or structured in a way that it is not necessary to compare all fields to find the best match. The disadvantage is the dependency of the implementation of the saved situations. How the fields are compared and stored should be changeable and it is a problem if the fields must be sortable. For example, the currently used raster is not sortable.

Another possibility to reduce the comparisons would be to compare only a subset of the situations with the current field. If an unambiguous match is found in this subset, it can be used. Otherwise further situations have to be compared. This approach is realized by a loop (compare appendix A figure 16).

8.7. Class diagram

The showed class diagram shows the structure of the code. One important architectural principle is the exchangeability. All members of the Tigers Mannheim are students of the DHBW. So they can participate for only 3 years. This leads to a high fluctuation in the team. Different generations of team members want to try different approaches. Especially the play finder is a module which is under change all the time. Before it was started with the learning play finder there were already two play finder. The first generation is a point-based play finder. Every play gets points according to the current field. This play finder did not worked very well so it was replaced by a very simple play finder, the basic play finder. This one reacts only on the referee commands. If there is no command, the bots try to get the ball. If they have the ball they shoot it in the direction of the goal.

By implementing the third play finder it is clear that it will not be the last or at least that the play finder will be modified a lot. Therefore the first challenge is to create a play finding system where the play finder itself is exchangeable and all major components like the knowledge base or the knowledge fields can be exchanged. This can be done by interfaces. The class diagram shows, for example, that there are already two implementations for the knowledge field. The bot assign and the raster implementation. It can be easily switched between this two without changing a lot of code. Knowledge field classes have to override only the compare method.

The reactions to the referee commands implemented in the basic play finder were added to the abstract class APlayFinder. On this way all future play finder can use this methods and do not need to implement this methods again. The entry point to the new play finder is an Adapter because the modules use the three methods choosePlays, reactOnRefereeCmd and reachtOnSituation as interface to the play finding module. This methods were translated to the match the new play finder.

9. Ideas for the future

9.1. Point system

At the moment every play can only succeed or fail. However, in a lot of situations you cannot really determine if it was successful. For example, if we try a direct shot and it results in a throw-in or in a free kick for us the algorithm would save this as a fail. Only a goal would be a success. However, having a free kick is much better than nothing.

A point system could be for example:

- 10 pt for a goal
- 3 pt for a free kick
- 1 pt for a throw-in
- -1 pt for a throw-in of the enemy

9.2. Improvement of the comparison algorithm

Despite a lot of time was spent into the comparison algorithm the solution is not satisfying. The current solution is not very good in performance and it does not regard patterns. Only fixed positions are compared.

9.3. Comparison with a series of World Frames

Next to the kind of the comparison algorithm the application of the comparison algorithm can be improved. If new plays are needed, the plays are chosen on the basis of the current field. However, it would be much better to decide it on a series of plays. On this way the knowledge base is compared to the field which will be established in the next milliseconds and not with the current field. Thus, we could change our plays earlier and react faster on new situations.

9.4. Knowledge fields in a database

To improve the comparison and especially the save and load process of the knowledge base it should be stored in a database. At the moment, XML files are used to store the database. It takes a long time to load and save this.

9.5. Changeable acceptable match

A play is chosen if there is an entry in the knowledge base which has a comparison value of 0.8 or higher. This is very static. This value should be variable. First, it should be changeable from a config file. In the future it would be nice if the value is changed dynamically. If the Tigers are winning this value should be high. This would result in a very conservative play. If the value is low the plays are chosen more creatively and randomly. This would be good if the Tigers are losing. On this way we would try new plays. Hopefully, this improves the situation.

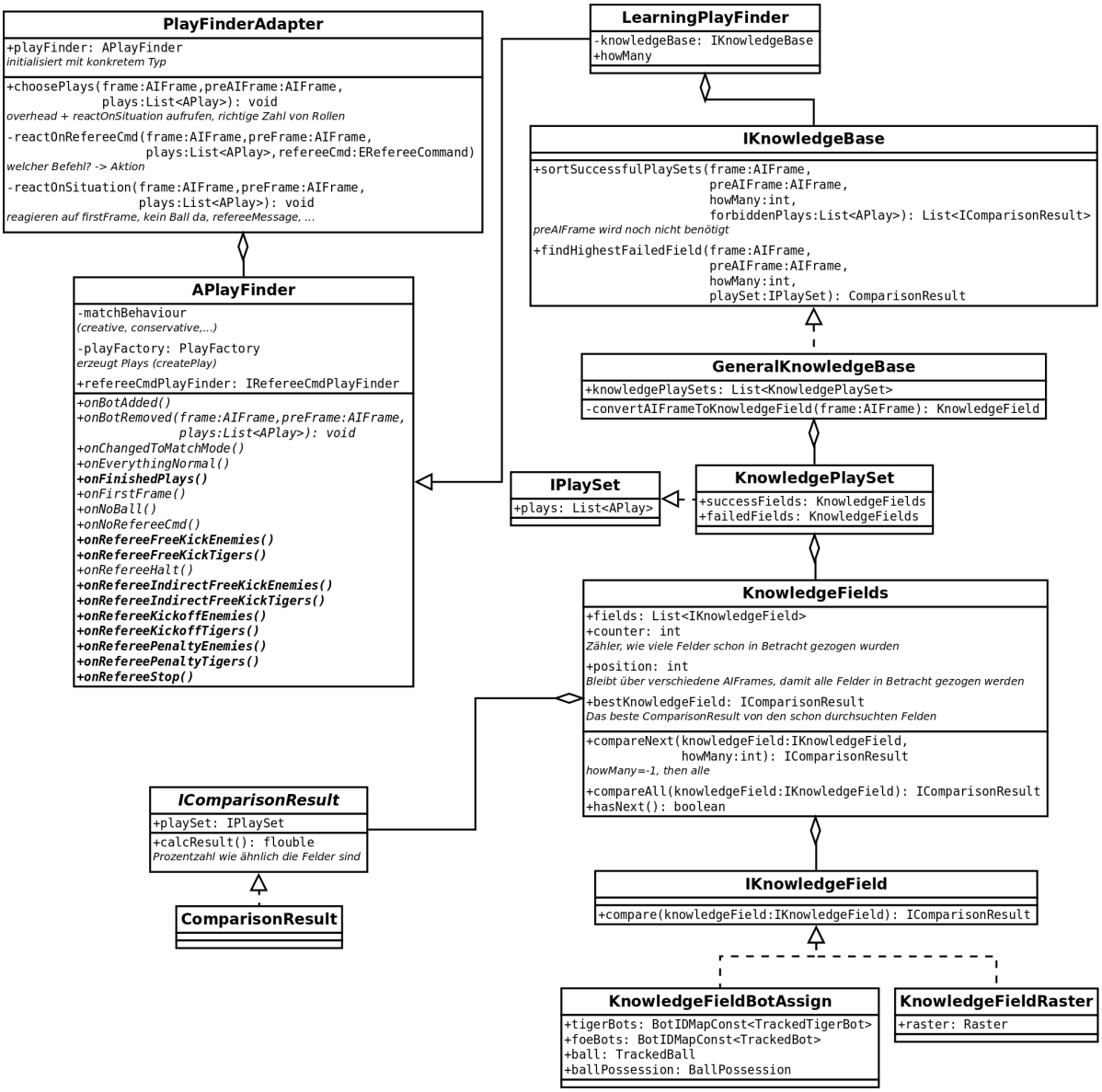

Figure 10: Class diagram of the learning play finder component

Part III.
Project Offensive Play

10. Introduction

The second part of this paper wants to design and implement a strategy to shoot goals. Thereby the strategy must be tricky to outsmart the defensive of the foe. In the following sections the strategy wants to be explained, the problem will be analyzed and the play-role pattern will be introduced. Additionally the system wants to be compared with the concept of the goal based agent. At the end the implementation of the strategy will be explained.

11. Play-Role Pattern

11.1. Basic aspects of a play

The basic aspects of a play can be transferred from the real soccer: everything is based on the position and roles of the players, as well as the ball possession. An example: whenever the enemy (e.g. red) is in ball possession and trying to score a goal, the other team (e.g. blue) will try to capture the ball while defending their own goal. When inspecting all kinds of situations, separations of plays can be achieved into three different categories:

- Defensive Plays (e.g. marking forwards, defending the goal)
- Offensive Plays (e.g. shots, passes near the goal area)
- Set Piece Plays (e.g. corner kick, throw-ins)

Those aspects can be (and are) transferred to roboter soccer. But unlike in real soccer, the bots need to be told what their roles are and what they're supposed to do. The instructions for the bots are covered within a play.

11.2. Aim of a play

As mentioned in the previous section, the bots need to be instructed what they're supposed to do while having a determinable, evaluable outcome. This serves as aim of a

play. Each and every play has different set of instructions, as well as (internal) goals. For instance: a defender bot is told to block possible shooting paths of the enemy while maintaining a solid defense structure (which thus pressures the enemy). A forward, however, is told to shoot goals. All of those instructions are evaluable in terms of success. In other words: whenever a play is executed, the wanted and determinable outcome enables an evaluation of said play.

11.3. Concept of the pattern

During a game different plays can be executed at the same time. Thereby each play follows an own strategy and consists of one to six roles. Each role is directly mapped exactly to one bot. Figure 11 wants to illustrate the relation between the Sumatra view with a play and roles and the hardware view with bots.

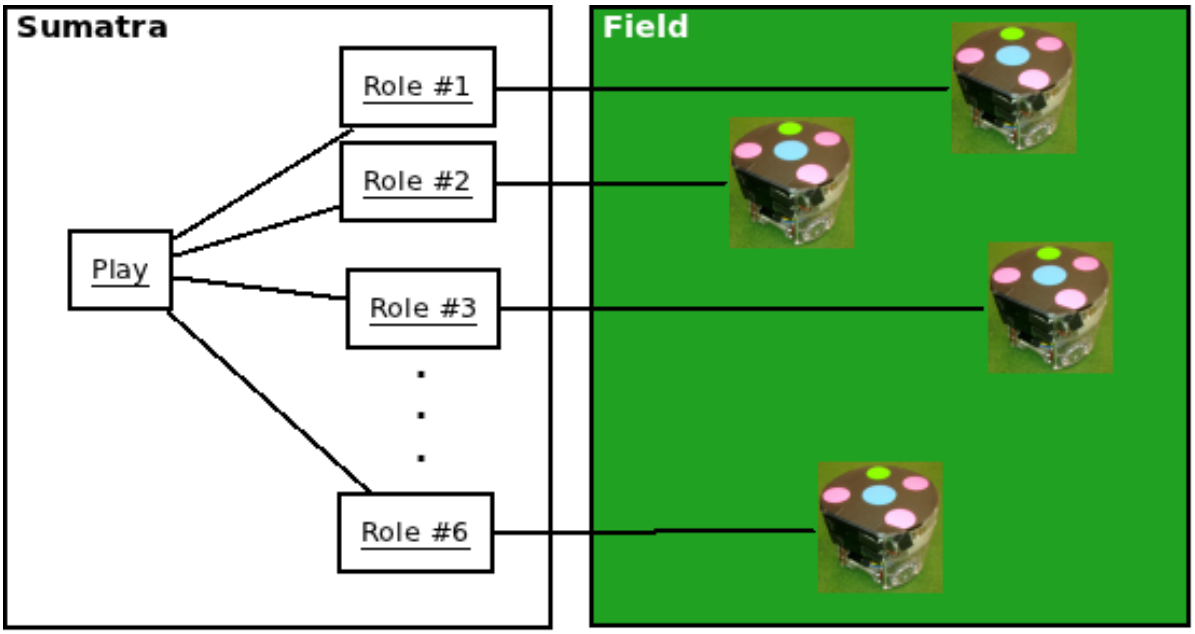

Figure 11: Relation between play, roles and bots

Each role just knows its own task and does no't know the other roles. Only the play will know all roles and will handle interactions between the roles. Figure 12 shows the implementation of the play-role pattern within Sumatra.

Abstract classes like APlay and ARole describes the basic structure for all plays and roles. Each play consists of one or more roles. The function "update(AIFrame)" hands over a AIFrame with all information about the current filed state to the Play. Information

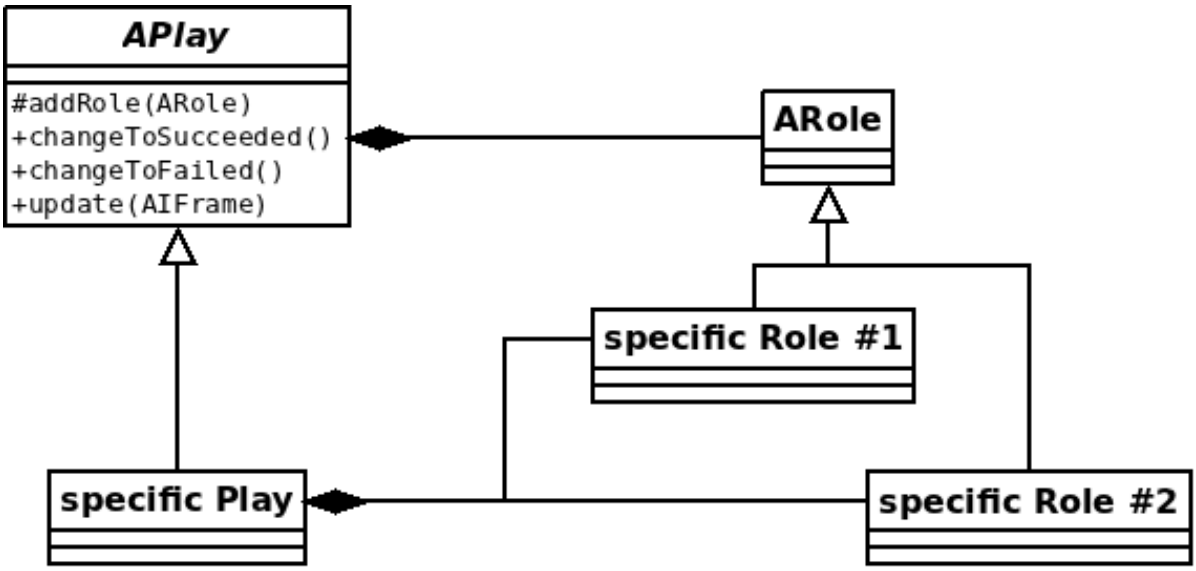

Figure 12: Implementation of the play-role pattern

of the AIFrame could be the position of the ball and the bots.

Further information and concepts will be discussed in [4] and [5].

12. Strategy of the offensive play

The strategy of a play can be categorized into offensive and defensive strategies. This project wants to develop an offensive play, which should be effective against a specific defensive strategy of the opponent. The defensive play of the opponent may consist of one or two defender and one goalkeeper. In case of a possible direct shoot toward the goal, the defenders and the goalkeeper will be placed between ball and goal. If there is no possibility of a direct shoot, the defenders may do not move their bots or another defense play will be executed by the opponent.

The idea of the offensive play is to hide the possibility of a direct shoot. The following steps want to describe the strategy of the offensive play:

Initial situation In the initial situation for this play, our team (red) got the ball, but there is no chance to shoot directly (marked by the red area at the following figure). But there is another bot of our team, which has a clear line of shoot, marked by the light green area at the following figure.

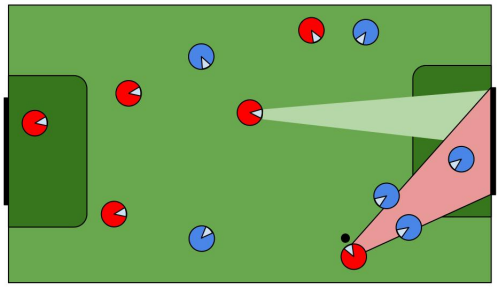

Figure 13: Play strategy: initial situation

Step 1 and 2 The basic idea is to pass to the other bot with the clear line of shoot and to perform an indirect shoot. But the most defensive plays are well-engineered, so that an indirect shoot could be parried. To perform an indirect shoot anyway, the basic indirect shoot play will be extended by a third role. This role want to block the own team bot with the clear line of shoot. This movement is shown in the figure below. The advantage of this movement is that the opponent team may don't recognized the new situation as a dangerous situation, because there is no direct shoot possible. Furthermore this "blocker" bot want to reserve the clear line of shoot and the opponent team is no't able to move their bots into the line of shoot.

After the "blocker" bot is placed the bot with the ball will be pass to the other bot.

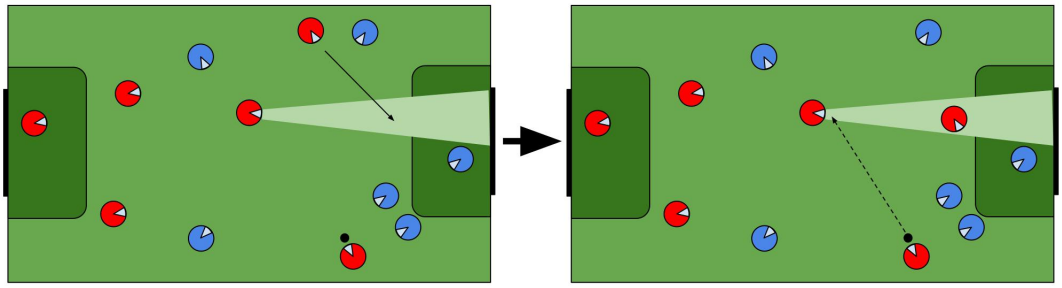

Figure 14: Play strategy: first and second step

Step 3 and 4 After the pass the bot, with the own blocked line of shoot, should got the ball. The new situation is shown in the figure below. Now the "blocker" bot has to move away to make clear the line of shoot. Directly after this movement, the bot with the ball want to shoot directly toward the goal. Once the bot preformed the shoot the play will evaluate whether or not it was successful. The result will be returned to the play-finder.

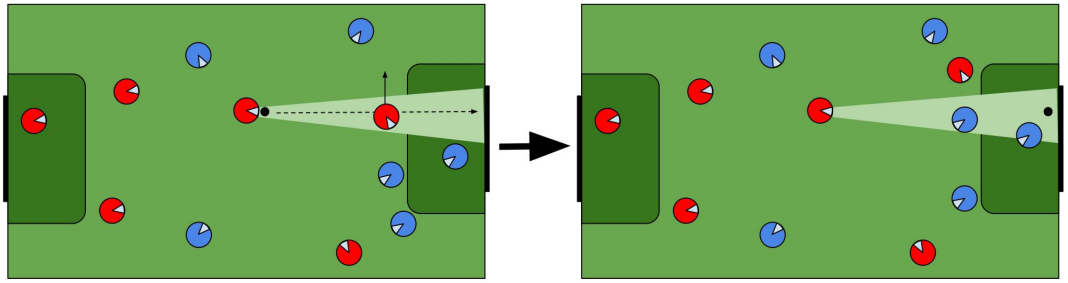

Figure 15: Play strategy: third and fourth step

13. PEAS Analysis

This project is based on the same circumstances as the project 'play finding'. Therefore the PEAS analysis will be the same and do not cover any new aspects, so it will be skipped at this point and referenced to the chapter 6 PEAS Analysis.

13.1. Environment Analysis

To show the complexity of executing an successful play and to make the challenges visible, the environment of the agent shall be described.

Keyword		**Description**
Observable	Yes	The field with position of the bots and the ball possession is fully observable.
Deterministic	No	The actions of the foes are not predictable. The way the ball moves, e.g. when bouncing back, is not predictable.
Episodic	No	A decision can effect all future decisions.
Static	No	The environment is changing during the whole play. The agent does always has to observe the actions of the foes
Discrete	Yes	The occurring states, percepts and actions are well defined. Only through strict definitions highly complex situations can be managed

continued on next page

Keyword		Description
Single-Agent	No	A multi agent environment exists. The own team can be seen as one agent, but the foes have to be treated as separate agents

Table 1: Environment analysis

14. Agent type

For the execution of plays a goal based agent is used. The goal based agent is based on the model agent (see [6, p. 17]). Model agents are able to handle an observable environment. The world is build as an abstract model and the agent has full access to the information. In our case, the abstract model of the world is the world frame. The world frame contains the position of all agents and the ball possession. The field itself is observed by cameras. The specific rules or conditions control the agent. A distinctions between aborting and further action rules has to be made. Aborting rules lead to a stop of the current play an give control of what to do next back to the play finder.The problem is to decide, when a play should be aborted or when the play is still executed and responsible to find a solution.

The difference of the goal based agent to a just model driven one is that he is aware of an desirable situation. So a kind of solution finding and planning of actions becomes necessary. In a complex scenario like robot football there are at least two possibilities to make decisions. The agent handles as many situations as possible or does a quick abort and the play finder has to handle the situations. No evaluation of the environment is done and the agent just follows defined instructions.

Because there does not seem to be a best practice, the decision was made, to let the agent evaluate what action seems to be the best in the beginning of the play and execute. If the planned actions fail, the play is aborted. The conditions for an abortion are turnovers or if there is no possibility to shot at the goal after a specific amount of time.

15. Possible Problems

Due to the fact, that the play is executed in a dynamic environment - allied and enemy bots move all the time - it is likely that the play itself will not always be successful.

One possible problem is the reaction time of our own bots: During runtime, a lot of decision have to be made by the agents which of course needs time and an informational foundation. Once done, we need to transfer said decision information to our bots which again need additional time to execute and "react". If the delay spikes (e.g. 250ms instead of 20ms), the previously decided actions may be obsolete again or cause the bot to not be at the wanted position in time. The intelligence of the enemy bots is another possible problem. Depending on how intelligent their agents are (e.g. whether they only let their keeper face in the direction of the ball in relation to the goal or they check if a bot cannot possibly shoot due to blocks), our play can be either very successful or failing quite often. A counter-measurement is choosing the play a few times and let our play-finder decide if the play has a possible way to be successful. In case it decides it was not successful after a predetermined amount of times, we rather focus on other plays instead.

16. Phases of the Play

The created play can be divided into three parts, all of which are equally important:

- The preparation part
- The main part
- The termination part

All of the phases will now be described.

16.1. Preparation part

In order to set our play in motion, preparations need to be made. Like in real soccer, the moves of our bots need to be well prepared, otherwise the play is likely to fail.

One of the preparations lies within the role assignment: depending on their current positions, bots need to be chosen which are most likely to execute the play successfully without too much "overhead work". For example, it is not reasonable to let a blocker cross the entire field in order to block our shooter. A higher success rate can be achieved by selecting a bot right next to our shooter as a blocker.

As for the previously mentioned positions, it is also important to check how strongly our bots are covered. This leads to the next preparation: reasonable positioning. Whenever the play is executed, it is of high importance, that the shooting bot has plenty of

room around it to not be blocked during the shot. In case that the bot is currently marked by another, it is supposed to find an area (within a reasonable radius) in the knowledge base which is not as strongly covered in order to maximize our possible success chance. Once these preparations have been done, the main play can be executed.

16.2. Main part

The main play itself concerns only the execution of the predetermined sequence. If the play is able to get to this point - whenever the preconditions are not met, the play won't be executed further - our play is likely to be successful. Once this point is reached, the bot which is currently in ball possession is to pass the ball to our shooter. Of course, this only applies if there's a pass path which ensures a high likeliness of reaching our shooter. In case the ball is halfway through, the blocker is told to unblock the shooter in order to free the shooting path. The reason to not immediately tell our blocker to leave the shooter is to ensure that the enemy AI won't cover the shooter or its shooting path in time which would then render the play itself useless. This also applies to unblock the shooter in "last second" fashion: if our connection to the bot spikes in this very moment, the shooting path will be blocked by our very own bot.

After the blocker freed the shooting path, the termination play is executed.

16.3. Termination part

Unlike the parts before, this part only covers the shot of our shooter. The reason to split this part from the main play is that everything before was a setup for the shot and, once done, the play is finished.

The ball was passed to our shooter and the blocker unblocked the way. Once the bot received the ball, it aims for the goal and shoots. Afterwards, the play is terminated and, based on the outcome of the play, other plays are chosen.

17. Ideas for the future

This sub chapter will provide information and thoughts what the developer team plans for future releases of the Currently, static positioning is used due to the fact that (useful) positioning itself in a dynamic environment is complicated. As for a future release, it is planned to analyse the field and generate positions which enable our play to have a higher success rate. In order to accomplish this, every position of the bots need to

be mathematically evaluated and, based on these information, choose the best solution. Of course, this analysis is done by the system in regular cycles with a sophisticated algorithm.

As explained in chapter 15 Possible Problems, the timing between the passer, shooter and blocker is very important. A plan for improving the timing among the bots is another idea for a release in the future. Due to several circumstances inside and outside the play, it is hard to implement a reliable timing among the bots (for the time being). Once these problems are solved, the improved timing will be added as another potential increase.

One of the last ideas is that, instead of just statically stand in one position once it's chosen, the bot and its blocker can dynamically move around the field where loopholes are given. This, on the other hand, requires a very high aiming precision for the pass and shot, as well as a properly working field analysis. After these factors have improved - virtually and physically -, the play will be improved as mentioned before. This will lead to a more complex situation for our enemy which will be hard for them to handle.

Part IV.
Summary

18. Conclusion

The project learning play finder has discovered how an agent for selecting several plays for the central control software of Tigers Mannheim RoboCup project can be implemented. Existing play finders are static and not efficient enough, therefore a new concept was developed. With this approach the agent will compare its observation with previous ones and will determine, which previous situation was the most similar one, what decision was with this previous one was made and the feedback of this decision. These determination will decide based on its knowledge base the best decision and decide it again, after the decision has a result, it will be saved in the knowledge base. There are several ideas for comparing different situations and one is used due to different issues.

This project has a big impact to the performance of the team Tigers Mannheim, because a play finder determines how the bots play together and that is an important part for winning a game.

The project offensive play showed how important knowledge based systems are. Depending on the given information, the play can be either successful or failing: existing plays were not as reliable on information as this play which discovered a few problems which will be fixed in the future. For instance: inaccuracy of our own bots as well as possible timing issues due to lag spikes. Once the problems are mastered, future releases can be tackled. Additional functionalities, such as dynamic positioning based on a field raster evaluation, can be implemented as well.

The conclusion is that the project helped to discover a few flaws in our system which will be fixed and expanding our play repertoire. This will increase the Tigers Mannheim's overall competitiveness.

Both projects will improve the performance of Tigers Mannheim, since there is now a new intelligent play finder and a non trivial offensive play, which can confuse the enemy software agent.

References

[1] König, C. and et al., Artificial Intelligence, Technical report, Tigers Mannheim, 2009.

[2] Vossen, P. H., Lecture 7 - learning agents, Presentation Knowledge Based Systems, 2012.

[3] Steinbrecher, O. and Birkenkamp, P., Taktische Spielfeldanalyse im Robocup mittels Rasterung des Spielfelds, Technical report, Tigers Mannheim, 2012.

[4] Coradeschi, S. and Karlsson, L., *A role-based decision-mechanism for teams of reactive and coordinating agents*, Springer-Verlag, 1998.

[5] Igarashi, H. and Kosue, S., *Individual tactical play and action decision based on a short-term goal*, Springer-Verlag, 1998.

[6] Vossen, P. H., Lecture 2 - intelligent agents, Presentation Knowledge Based Systems, 2012.

Glossary

Worldframe The Worldframe is a data container with the data about the bots. This informations are fetched from the vision and directly from the bots. It contains, for example, the position and the orientation of the bots.

AIInfoFrame The AIInfoFrame contains further information about the field. It contains the Worldframe and next to it data like the ball possession. This information is calculated in Metis.

Bot A bot is the physical robot on the field. In Sumatra the robots are named bots.

Play A play is a global strategic instruction set for the bots. Plays are divided into offensive and defensive ones, as well as set pieces.

Role A role is assigned to a bot within a play. It enables the consequent usage of a bot for a play while giving the bot an internal instruction set.

Knowledge base The knowledge base contains information which was learned from former games. It contains knowledge fields.

Knowledge field A knowledge field contains a play situations (bot and ball positions), the play which were chosen in this situation and the result of the play.

A. Play finding

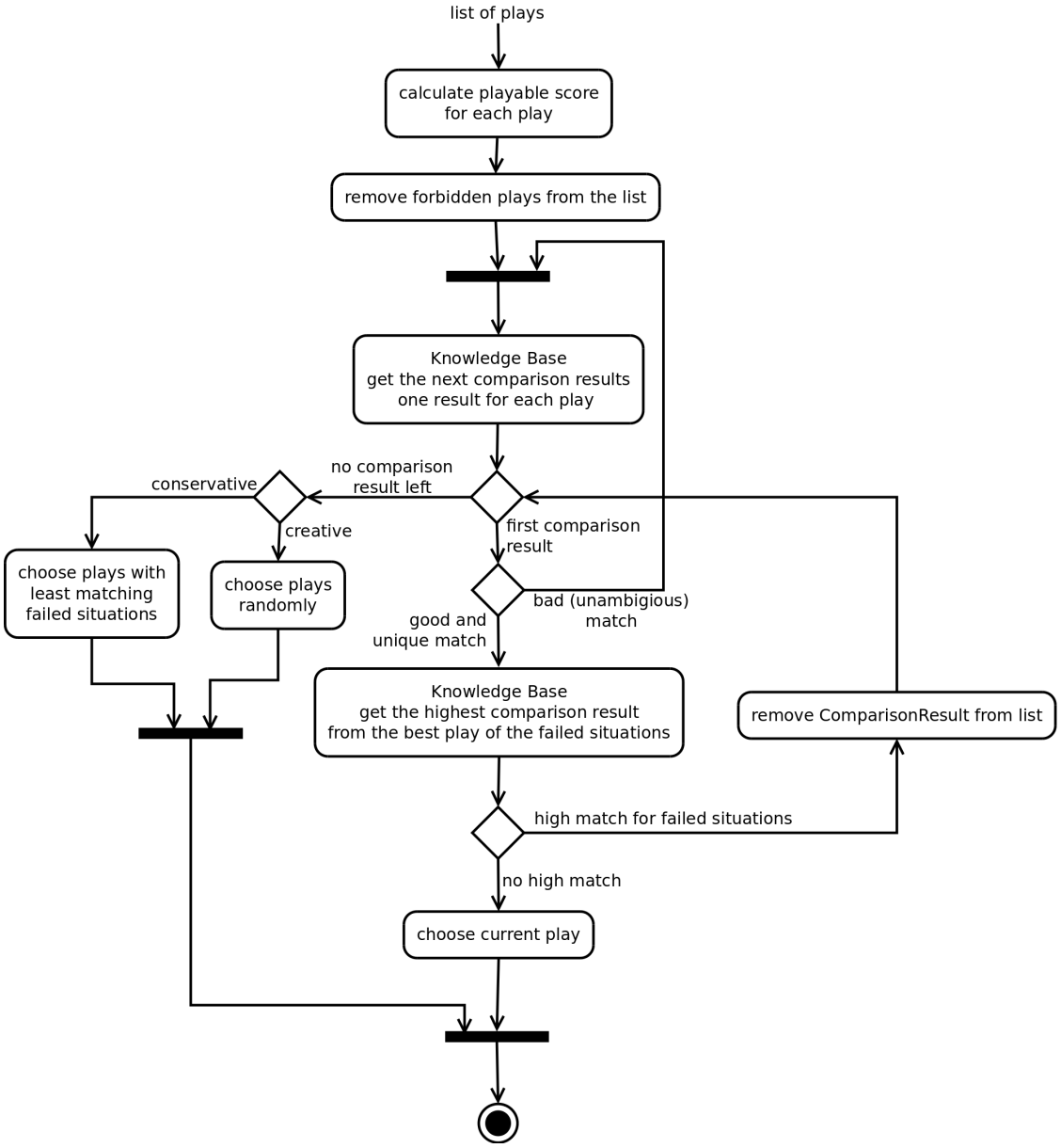

Figure 16: Activity diagram of learning play finder component