BW Cooperative State University Mannheim, Germany

Study Report

AI Architecture and Standard Game Strategies in RoboCup SSL

Architecture of game logic and standard implementations for the AI of TIGERs Mannheim

Name:	**Nicolai Ommer**
Matriculation:	2573200
Course:	TAI10ABC
Study course:	Applied Computer Science
Study manager:	Prof. Dr. H. Hofmann
Tutor:	Prof. Dr. J. Poller
Semester:	5. - 6. Semester
Date:	06/03/2013

Ehrenwörtliche Erklärung (Declaration of Academic Honesty)

Gemäß § 5 Abs. 2 der Studien- und Prüfungsordnung DHBW Technik vom 18.05.2009 versichere ich hiermit, die vorliegende Arbeit selbstständig und nur mit den angegebenen Quellen und Hilfsmitteln verfasst zu haben.

06/03/2013	
Datum	Nicolai Ommer

Abstract

In the RoboCup SSL one of the key challenges are good strategies for coordinating all robots with the aim of scoring goals. The team TIGERS Mannheim uses a central software called Sumatra which has a good base for creating and choosing good strategies. There are, however, some drawbacks in the architecture and Sumatra is still missing some good strategy implementations.

Many developers worked on the strategies and many left the team afterwards. Nobody had an overview of currently active plays and their status. With this report, the whole AI architecture and all implemented strategies, called plays and roles, were analysed and optimized. The aim was to get a complete overview of all strategies and make it easier to hold the overview within the code.

Another aim was to make the development of plays better understandable by creating a documentation and clean up and document the code of existing plays.

The new architecture enables more flexibility, less redundant code and better understandability. Deprecated plays and roles were removed and all existing plays tested and documented. Several features were added to the GUI of Sumatra to help developers with testing their work.

Contents

List of Figures

List of Tables

Listings

1 Introduction

The following sections will explain what the RoboCup SSL and the team TIGERs Mannheim is. Furthermore, drawbacks from last year and aims for this year will be summarized.

1.1 RoboCup SSL

The RoboCup is a competition where robots compete against each other. It was founded in 1997 and consists of several different leagues. The overall target of the RoboCup is defined as follows:

By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, complying with the official rule of the FIFA, against the winner of the most recent World Cup.[1]

The most common league is the standard platform league where small humanoid robots, called NAOs, are used. This report is based on the Small Size League. There are six against six small cylinder like robots playing on a 6x4m field. They are detected with cameras that are mounted on the top of the field. Each robot has a unique color pattern that identifies a robot. The camera data is processed by a central computer and position data of both teams is broadcasted into a network. Each team has a central computer which receives the data, processes it and sends commands to the robots with a technique of their choice. The robots are build by the teams. While their are some standard idioms in design, teams are free to design their robots as far as they comply to the rules.

A match is two times ten minutes long. As of 2012, the referee is a human and has no auxiliary. Referee software is under development [2] and likely to be used in the next years.

1.2 TIGERs Mannheim

The team TIGERs Mannheim was founded in 2009 by a group of students at the DHBW Mannheim with the aim of participating at the RoboCup SSL. It is divided in several sub teams. There is a team for the hardware (electronic and mechanics) a team for software (AI, infrastructure) and for marketing.

The team first participated at the RoboCup 2011 in Istanbul and a year later in Mexico City. The next RoboCup takes place in Eindhoven.

For Eindhoven, the team designed new robots with more precision and a chip kicker [3]. While the software performed good and mostly correct last year, the precision, intelligence and reliability must be optimized. The aim for the software for this year is to have more stability and reliable components instead of many insufficiently tested components.
One important new component that is already finished is the learning play finder [4].

2 Fundamentals

Sumatra is a complex software that integrates AI, robot control, visualization, etc. in one software. For this report, the AI with its Play/Role concept is important to understand the new features that were implemented. This section will explain some basics about the AI in Sumatra.

2.1 Play/Role/Skill Concept

Sumatra uses a classical concept for execution of tactics. It is based on [5], but was modified in some cases. [6] describes the AI in more detail. Basically, a role defines the behavior of a single bot. This could be for example a shooter role that calculates a shoot target, aims and shoots. The actions are defined using skills. A role is mapped to a specific bot for the life time of the role, which is usually around some seconds. Plays are used to coordinate multiple roles, for example a shooter and a receiver. There are roles that can be used universally in different plays, like simple BallGetters, and there are more specific roles that are only used by one play. The more general a role is, the better it can be reused for new plays.

There are usually about three or four plays active at a time. One is for dealing with the ball, one is for defending the goal and the rest is for supporting or preparing bots. Plays are changed very frequently. Each play has the possibility to finish. Usually, only offensive plays which are dealing with the ball will finish themselves. A defensive play will just defend all the time, because it has no short time goal.
As soon as one play finishes, the play finder will select a completely new set of plays.

A skill is a high abstraction of an action that should be executed by a robot. It is the bridge between AI and robots. The AI can create a skill for moving to a destination while looking at a target. The skill will transform this into commands for the robot which is basically the speed for each motor. Skills are executed by the skill executor which will periodically send updates as commands to the robots.
The new robots will behave differently. Basically, they receive a spline and do the motor control on their own.

An important fact to keep in mind is, that a RoboCup SSL match is very fast. There is not much time for preparing or second tries. Plays should be designed to efficiently

executing their task and if something goes wrong, give up and let the play finder find a new play.

2.2 Conditions

Conditions are used to check if a desired requirement is fulfilled. There are conditions for checking if the robot reached its destination or looks to the desired target. Another conditions checks if the destination is free of other bots. Several conditions check the visibility from a bot to the ball or a target.

The idea of conditions is, to avoid duplicate implementations of the checks. Also, the result will be cached, so that no unnecessary checks are done.

2.3 Integration into AI Module

Sumatra is divided into several modules. Modules are named after Greek gods. The AI is controlled by a thread called Nathan. It grabs the most recent WorldFrame and creates an AiInfoFrame. This will be passed to the AI modules. Some will put information into it, others will read from it. The flow is shown in figure 1.

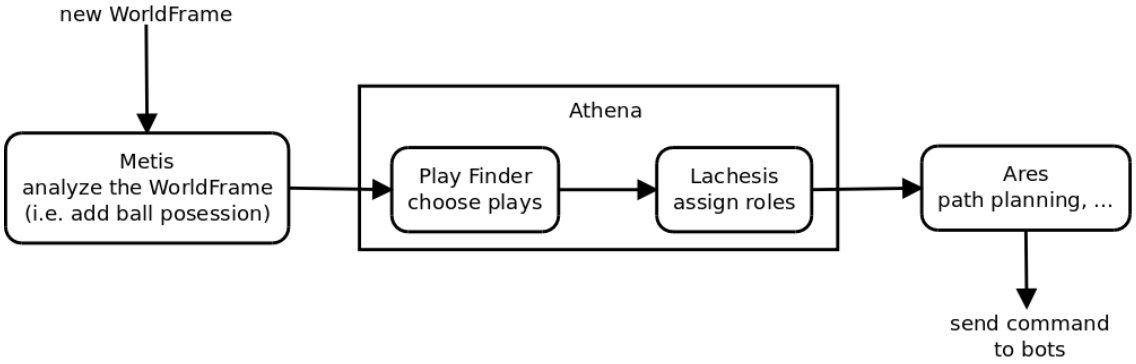

Figure 1: AI modules, processed by Nathan

The first AI module is Metis. It contains a set of calculators. Information like the current ball possession are calculated there and are stored in the AiInfoFrame. Next comes Athena which is responsible for play selection and execution as well as role assignment. Finally, Ares takes the skills from the currently active roles and gives new skills to the skill executor.

There are more modules, but they are included in those three main modules, like Lachesis which is responsible for role assignment.

3 Mandatory Behavior during a match

During a match, all robots have to behave on certain situations that are announced by the referee software. Additionally, there are some basic strategies that should be implemented and working properly. This section will explain the different game stages as well as some important strategies and details about the realization in the plays.

3.1 Game Stages

Game stages are announced by the referee software and teams have to behave according to the current game stage. Some stages are more important for the AI than others. An overview of all referee commands and the command flow can be gathered from figure 2. The game stages for the AI are a bit more specific and can not be mapped exactly to the figure.

After a normal start, teams are free to get and work with the ball. This is the most important game stage for the AI, because in this stage, there is a high flexibility and thus much potential for different sorts of plays.

A game starts with the kickoff game stage. The referee sends a signal telling which team has kickoff. The ball may not be kicked before a ready signal so that both teams have time to send the bots to their positions. The kickoff stage will pass into normal game play after one bot touched the ball. A kickoff is almost static. Teams may have different start positions and strategies, but in fact one play should be enough to deal with the kickoff.

Videos from RoboCup Japan Open 2013 finals[1] showed that chipping the ball straight to the opponents goal is a good way to start the game. Although the ball will most properly not reach the goal, it is already in the opponents half.

Next, there is the freekick game stage. It is used to bring the ball back into play after a foul. Depending on the foul, the referee will announce a direct or an indirect freekick. Direct freekick means that the shooter may directly score a goal, while on a indirect freekick the ball must touch another bot before. Freekicks are executed from where the offence occurred or from the field border in case of a throw-in.

The freekick game stage is very important for the AI, because the chances for scoring a

[1] http://www.youtube.com/watch?v=6BchV1Pk7yc

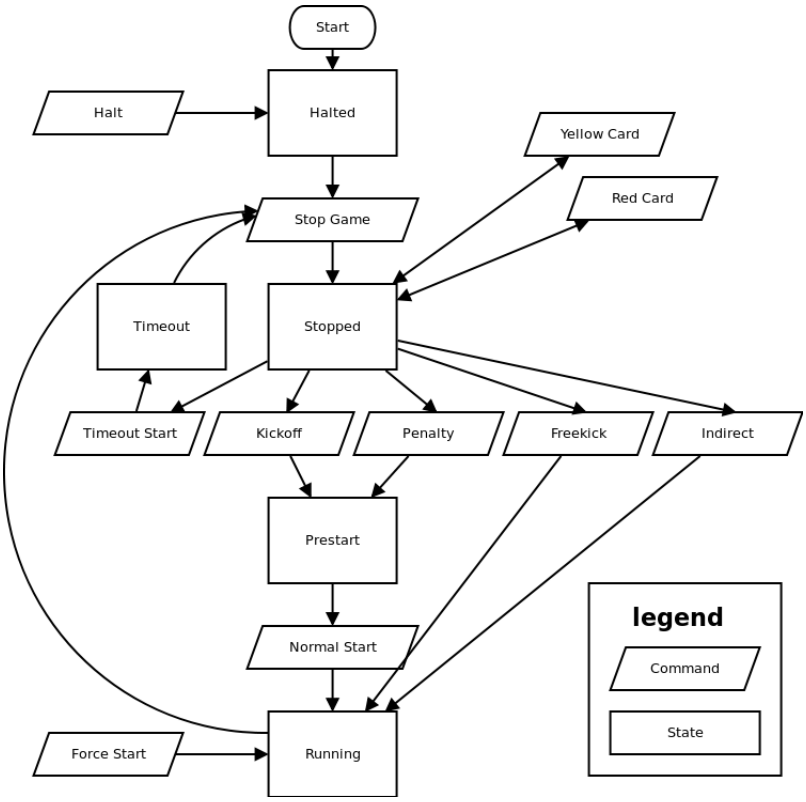

Figure 2: Referee commands and game stages of a RoboCup SSL match

goal are better than in the normal game stage. The AI has some more time to calculate an optimal strategy, like a good receiver that can score a goal with an indirect shot.

A less common game stage is the penalty kick. The ball will be placed on the penalty mark and the team that got the penalty kick can shoot directly on the goal while the opponent may defend with a keeper. After the shooter kicked the ball, the game stage will pass over to normal game.
The only task the AI has, is to let a shooter role find a good target and shoot. More important than the target is the precision and speed of the skill and hardware. The faster the bot can shoot, the less are the chances for the keeper to hold the ball. Same goes for the keeper. It has to move as fast as possible, while keeping an optimal position to minimize the way for blocking the ball.

Last, there are two game stages for stopping the game. The referee software will send a stop signal in case of a foul. All bots must immediately hold a distance of 500mm between the ball and themselves. In rare cases the referee will send a halt signal to signal the bots to stop completely.

3.2 Kinds of Plays

Plays can be put into different groups depending on their responsibilities and actions. During normal game play, there are three kinds of plays: defensive, offensive and support. There may only be one defensive and one offensive play at a time. The defensive play includes the keeper role and defenders that are synchronized with the keeper. The offensive play includes a bot that works with the ball and optionally one or more helper roles like a ball receiver. The rest of the roles are put into support plays. Those roles must not work with the ball. They can, for example, mark an opponent or find a good positions for later plays.

Beside the three game types, there is also the standard play type. This includes all plays that are only used in standard situations that were announced by the referee.

Plays can be created with a variable role number. They define how many roles they can handle at minimum and maximum. In order to avoid chaos, it is considered that a play may not be active twice. For example, if more than one marker is needed, all markers will be coordinated by one play. The PlayFinder module will care about the selection of plays and the number of roles per play.

3.3 Basic Strategies

There are several strategies that should be available during the normal game stage. They may be included in a more general or more specialized play or may be implemented on its own.

First of all, there needs to be a role that gets the ball. This should be separated into a single role, because it is needed by many different plays. Also, ball getting has some parameters, which especially depend on speed. The role may want to touch and/or dribble the ball or it just wants to keep some distance depending on what to do next. It may also drive directly towards the ball and optionally turning around it or drive around the ball in a wider distance. In the second case, the opponent may be faster at the ball.

Simply getting the ball is only possible, if the opponent does not have the ball. Thus, it is also reasonable to have a role that can conquer the ball. The dribbling device is able to pull the ball back. The bot can combine the dribbler with turning and moving

back. Finally, the bot should be between the ball and the opponent bot to avoid that the opponent bot gets the ball back.

When a bot has the ball, there are several possibilities. If the bot can shoot directly into the goal, it can do a direct kick by calculating the best target. Else, it can either pass the ball to another bot that has a better position or it can do an indirect kick with another bot, which means, the second bot receives and immediately shoots into the goal. The disadvantage of passing is, that it is very slow and the opponent can prepare the defense.

In general, the most important fact to keep in mind is, that a SSL game is very fast. Long preparation is not possible, because the opponent will have the same time to prepare against the attack. A simple straight pass is thus not reasonable in most cases. Instead, the shooter should kick the ball in a way that the receiver can use the speed of the ball and does not need to turn. The chip kicker can be useful in this case by chipping the ball over the receiver.

Last, it is always a good idea to prepare some bots for later plays. There should be at least one bot that moves to a free position from which it can receive a ball and score a goal.

4 AI Architecture

A good base architecture is the key to reliable and stable software. A change in the architecture often means that the overlaying code must be changed, so changes should only be made if necessary and as early as possible. The AI needed some improvements that are outlined in the following sections. One big change was the new skill concept for a better communication between roles and skills. Additionally, several smaller improvements were implemented.

4.1 Communication between Roles and Skills

Roles represent a certain tactic that has to be executed. The role may decide when to move where. It should, however, not deal with any special movement. This is the task of a skill.

Last year, there was a classical miscommunication between the developers of the skill system which is more robot orientated. Skills were divided into three groups, namely move, dribble and kick. Roles calculated new skills for each frame. In order to compensate the amount of commands send to the robot, the skill system filtered new skills by comparing them with the current ones and changed only skills that differ. With the current architecture, roles had no chance to create a more efficient implementation, because they were not able to get feedback from skills. So the complete architecture needed a redesign.

The first big change was to remove the skill groups. With the new robots, a new move concept was added that made it easier to implement move skills without path planning. So moves like turning around the ball are now easy to implement. Before, only few basic skills were implemented, but now there is much more potential in the skills. A role should not care about complex moves, it should only care about where to move. So there should be skills that do a more complex move combined with dribbling or kicking. The result is, all skills are based on the abstract move skill and have access to the dribbler and kicker device.

The central idea for the new architecture was a new dynamic state machine that must be used by all roles. This state machine should cleanly define single states and their transitions. Each state should be notifiable about skill completion.

The state machine is designed in a way that it can be reused. Figure 3 shows the class

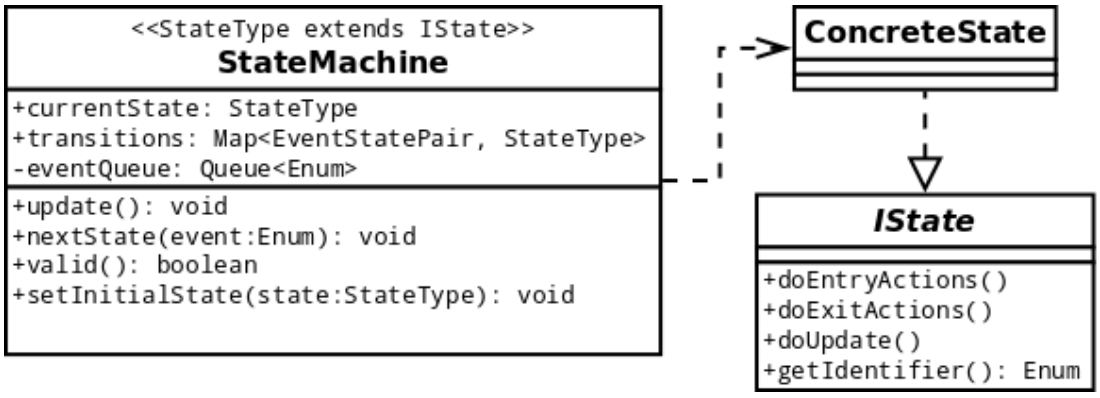

Figure 3: Class diagram of the new dynamic state machine

diagram of the state machine. The standard state defines methods for entry and exit actions and continues update. Thus it is possible to do some actions once the state gets active and before it stops. The update method can be used to check for some conditions during state runtime.
States are identified with an Enum that must be returned by getIdentifier(). Both, state identifier and events are declared as simple Enums. This makes the state machine more dynamic without the use of excessive generics.

A state machine has a map of transitions. The key of the map is build out of the combination of a state id and an event. The key maps to a concrete state implementation which is the state that should follow the last state with the according event.

State transitions are triggered using events. By calling nextState, the new event will be enqueued. The queue will be dequeued in the update method. This is illustrated in figure 4. Usually, the state machine is updated on each new frame. If there is a new event and a transition for this event, the current state will be stopped by calling exit actions and replacing it by the new state from the transition map. Entry actions and update will be called on the new state to initialize it.

The state machine is implemented into ARole which is the abstract class for all roles. Each role has to set an initial state and at least one end transition in its constructor. Without doing this, Sumatra will throw an exception stating that the role can not be created. A validation method in the state machine also checks if there is at least one valid path from the initial state to an end state.

States should be implemented as sub classes within the roles. An example is shown in figure 5. It shows three states for the BallGetterRole. The first state uses the

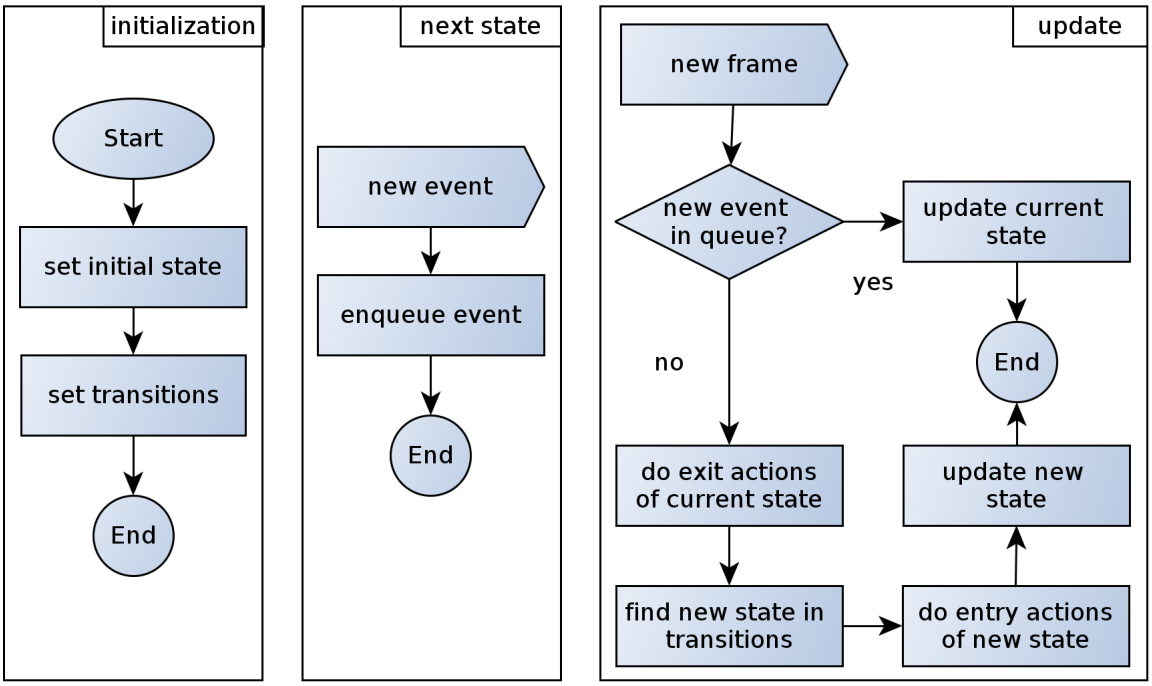

Figure 4: Flowdiagram of the internal state machine implementation

MoveToSkill to move near the ball. When the skill is complete, the state will get notified and can decide if the current position is correct. With an event, the transition to the next state is triggered. The GetState will touch and dribble the ball and the TurnState can than turn around so that the bot looks at the desired target.
The DoneState is a standard class that indicates, that the state machine is done. The role will be set completed, so that overlaying plays can decide what to do next.

4.2 Improvements for Play Development

Good, reliable plays are the key to success. Plays have to react on endless different situations. Thus, testing is very important. To enable developers testing their plays during development, there is a simulator software. However, simulator and real environment are often very different, so testing in the simulator is not sufficient.

An important part in testing is, to get enough feedback for verification. If something goes wrong, the error should be easily identifiable and reproducible. As the game is very fast, logging and recording sequences helps detecting wrong behavior.

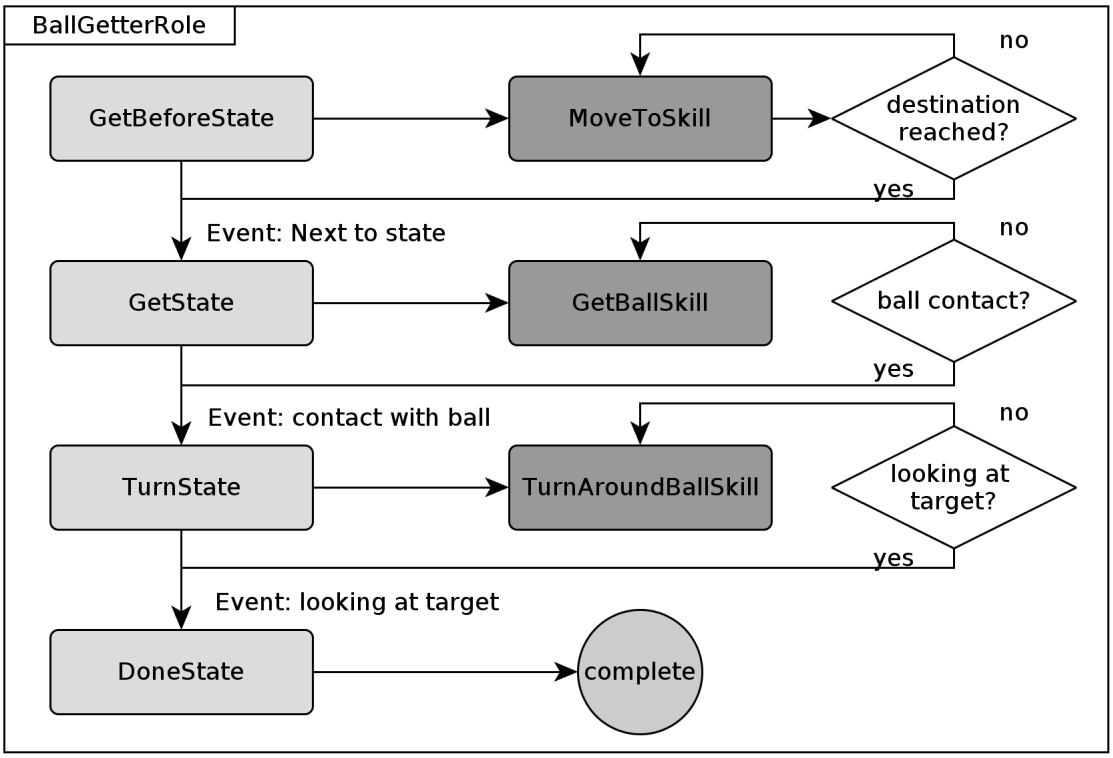

Figure 5: Implementation of the state machine in the BallGetterRole

4.2.1 Visualizer

To support play developers, several features were added to Sumatra.
Sumatra has a visualizer that displays the field with borders, bots and ball. With an option panel, additional information can be enabled.

The whole visualizer is now more flexible. It can be rotated, moved and zoomed in. This helps watching certain role behavior in detail.
Several layers were added, too. For example, there is a layer for showing the coordinate axis, a layer showing role names next to the bots and a layer showing distance marks for measuring shoot distances.

A more complex layer can display shapes, like simple points, lines, arrows and ellipses. They can be added within a play or role in each frame. This is an important tool for developers, because it is hard to tell if calculated vectors are correct. By visualizing results, they can be verified. Additionally, it is easier to understand what roles or plays are doing. The shapes may also be interesting during a match or general testing, to see what a role is doing and what it should do.

For testing and calibrating the chip kicker, it turned out that simply watching the ball and guessing where it touched the ground the first time is not easy. Therefore, a ball buffer layer can be activated that shows the last ball locations and highlights flying balls as red balls. This makes it possible to manually calibrate the chip kick strength.

Another feature is the record button. It will capture all AI frames until the button is pressed another time. All captured frames will be displayed in a new visualizer in an external window. The new visualizer also has the option panel, so additional layers can be enabled or disabled afterwards. The new window is independent from the main Window, so Sumatra will still work. It is also possible to record and play back another sequence and compare those two. In figure 6 the main visualizer and a record window is shown. For demonstration of the shapes, a testing play is active that lets the bot move in a rectangle around the center point while displaying several ellipses and lines.

Figure 6: Sumatra visualizer and record window

4.2.2 Play Management

Sumatra needs to handle several plays, developed by several people. Some plays work better than others and some replace old ones. The AI as well as the developers need to have an overview of all available plays and need to know what they are for.

Last year, the way, plays where instantiated was to use a factory with a switch-case over an an Enum for play identification. The Enum had to be set in the factory as well as in the play itself. In the factory, there was a huge switch block. This implementation worked and was fast, but it suffered the ease of maintenance and flexibility.

The aim of the new concept was to have a central place for management of plays and to avoid duplicated code. Plays should be put in groups and should be detachable. Also, it should be possible to instantiate them with a dynamic number of roles.

```
// example: Stop move play with at least one role and at most 4 roles
STOP_MOVE(EPlayType.STANDARD, 1, 4, StopMovePlay.class);

/**
  * Enum constructor
  */
private EPlay(EPlayType type, int minRoles, int maxRoles, Class<?> impl)
```

Listing 1: Constructor of EPlay. Each play has a type, min and max roles and an implementation

The implementation is a complex Enum. The constructor and an example is shown in listing 1. All plays are inserted into this Enum with a play type, the minimum and maximum number of roles the play is supposed to handle, and the implementation class. Instantiation of the plays is done using reflection. The only thing, play developers need to keep in mind is to have the correct constructor, shown in listing 2.

```
public StopMovePlay(AIInfoFrame aiFrame, int numAssignedRoles)
{
  super(aiFrame, numAssignedRoles);

  ...
}
```

Listing 2: Constructor of a play. Each play has to define exactly this constructor.

With those definitions, the play factory can instantiate all plays dynamically by invoking the one and only constructor using Java reflection. The implementation can be seen in listing 3.

The play type is another enum that includes the play types (standard, offensive, defensive and support) on the one hand and play states (test, deprecated, helper, defect, disabled) on the other hand. In order to stop the play finder from selecting a certain play, the only task to do is to change the type to something like defect.

```
public APlay createPlay(EPlay ePlay, AIInfoFrame curFrame, int
    numAssignedRoles)
{
  int numRolesToAssign = numAssignedRoles;
  APlay aPlay = null;

  if (numAssignedRoles == EPlay.MAX_BOTS)
  {
    numRolesToAssign = ePlay.getMaxRoles();
    if (numRolesToAssign == EPlay.MAX_BOTS)
    {
      numRolesToAssign = curFrame.worldFrame.tigerBotsConnected.size();
    }
  }

  final Constructor<?> con = ePlay.getConstructor();
  aPlay = (APlay) con.newInstance(curFrame, numRolesToAssign);
  aPlay.setType(ePlay);

  return aPlay;
}
```

Listing 3: Instantiation of a play in the PlayFactory

A documentation about how to develop plays is available in the Wiki [7].

4.2.3 Individual Robot Properties

In some cases it is reasonable to know some individual properties of a robot. Either for complying to the rules or for better robot selection. While it is basically possible to have different kinds of robots for different kinds of actions, this is not practical. Designing a single robot that can do all needed tasks must be sufficient.

However, practical experience showed that on long term, some robots are better than others. Some may shoot more precise than others because of mechanical defects. If a dribbler or kicker device is broken, but the robot can move and there are no exchange robots available, this robot could go into defense, where it basically only has to move. However, the AI has to know that a robot is missing some features when it assigns roles. Thus, a list of features for each bot was implemented. It is stored in the individual bot configuration and can be changed in the GUI.

Each role must override a method *fillNeededFeatures* which may look like in listing 4 to declare the features it needs to execute correctly. The RoleAssigner will compare the

```
@Override
public void fillNeededFeatures(final List<EFeature> features)
{
  features.add(EFeature.STRAIGHT_KICKER);
  features.add(EFeature.DRIBBLER);
}
```

Listing 4: Roles define which features they need for execution

available features of a bot to the required features of the role and add a penalty factor for assignment, if a feature is not available. Thus, other robots will be preferred.

There is another small robot property that is needed in some cases to comply to the rules. Some referee commands require that the robot that shoots the first time must not touch the ball a second time. In this cases, a temporary flag in the shooter robot indicates that it must not be selected another time, which is very likely otherwise when it is near the ball. This flag, however is irrelevant while the first role is still active. The role must check that it does not touch the ball a second time.

5 Analysis of required plays and their status

At the WCC in Mexico 2012, Sumatra had plays for all required situations. It could react on all referee commands with a suitable play and had play combinations for a running game.
However, the variety and intelligence has much potential for improvements. Some plays are quite old and the developers already left the team. Nobody knew the whole set of plays and could decide, which must be improved. The aim if this report is, to have a list of all plays with their status and to improve or fix plays if necessary.

This section will outline the status quo of the currently available plays in Sumatra and their status. It will furthermore go into more detail for some plays.

5.1 Status Quo

Sumatra had lots of plays some of which were not even used. In the context of this report, all plays and all classes were analysed and tested. Old code was removed and non working plays documented as a bug in the projects reporting software. The plays that were left over are documented in the following tables. Plays are distinguished between offensive, defensive, support and standard plays.

Play	**Roles**	**Description**	**Number of roles**
BallGetting	BallGetter	Get the ball by driving to it and try to conquer it against an opponent. Will pull ball back in an elliptic curve in order to pass ball or (if this play only has one role) be prepared to pass to a bot.	1
BallBreaking	BallGetter, TurnAround-Ball, SimpleSin-gleShooter	This play should try to break the ball clear. For this, the ball tries to get the ball and shoot it into the middle of the field without looking for a target.	1
BallConquer	BallGetter, BallConquer, PassSender, PassReceiver-Straight	Conquer the ball by pulling the ball back from the opponent and passing it to a free bot	1-2
DirectShot	Shooter	Find a good target on the goal line and shoot directly into the goal.	1
IndirectShot	IndirectShooter, Receiver	The shooter passes the ball to the receiver, the receiver redirects the ball directly into the goal.	2
Passing	PassSender, PassReceiver-Straight, Ball-Getter	This Play will pass a ball from one bot to another. The play will try to find a receiver for it self (by setting the init-Pos appropriate) The receiver will take the ball on the dribbler	2

Table 1: Offensive plays for a match in Sumatra

Play	**Roles**	**Description**	**Number of roles**
KeeperPlus-NDefender-WDP	KeeperSolo, BallGetter, PassSender, DefenderKND-WDP	Defense play that handles a keeper and all defenders. It uses calculated defense points for positioning and coordinating keeper and defenders. Additionally, it lets the keeper pass the ball out of the defense area.	1-4

Table 2: Defense plays for a match in Sumatra

Play	**Roles**	**Description**	**Number of roles**
BreakClear	Move	Let a bot break clear. The current approach is to choose a target randomly while keeping a distance of 300mm to any obstacle.	1-2
ManToMan-Marker	ManToMan-Marker	Each role of the play marks a dangerous opponent. The play decides which opponents are dangerous by checking several constraints.	1-3

Table 3: Support plays for a match in Sumatra

Play	**Roles**	**Description**	**Number of roles**
FreeKick-Marker	ManToMan-Marker	Marks dangerous bots during freekick preparation	2
FreeKickMove	MoveWith- DistanceToPoint	Stand between ball and goal to block direct kick to our goal. (keep distance to ball)	1-3
ThrowInUs	BallGetter, PassSender, PassReceiver-Straight	Do a throw in by finding a good receiver which can score a goal (implementation is pending)	2
KickOff- Indirect	Move, PassSender, Receiver	Do the kickoff by doing an indirect shot on the goal	2
PositioningOn-KickOffThem	ManToMan-Marker	Position bots for an kickoff by the opponent team while marking dangerous opponents.	1-4
PenaltyThem	KeeperPenalty-Them, Passive-Defender	Play is chosen when opponent team gets a penalty. We have to defend our goal.	all
PenaltyUs	PenaltyUs-Shooter, PassiveDefender	A shooter does the penalty while all other bots stand on a valid position (according to the rules)	all
StopMarker	ManToMan-Marker	Our bot will block one opponent pass receiver.	1
StopMove	Move	n bots will form a block in a distance of 500 mm to the ball in order to protect the goal after the game is stopped by the referee	1-4

Table 4: Plays in Sumatra for handling certain referee commands

5.2 The BallGetter

The BallGetterPlay is one of the oldest plays in Sumatra, because before a bot can work with the ball, it has to be near to it and possibly needs it on the dribbler. Most offensive plays will just use the BallGetterRole internally to ensure that the ball is in front of the ball. The use of the BallGetterPlay is only, to ensure that only we are in ball possession. Else we would need some ball breaking or conquering play first, because an offensive play like the DirectShotPlay will not deal with any opponents that also try to get the ball.

In practice, the game is so fast that any precision in getting the ball would take too long. The only task the BallGetterPlay should do is drive near the ball without any other constraints like reaching a target to look at. This is implemented by periodically checking the ball possession. As soon as we have the ball possession, which means we have a bot near the ball and the opponent has not, the play finishes. The next play can use a BallGetterRole with desired precision.

Robots have a dribbling device for keeping the ball in front. It is, however, not wise to use it too often. Getting the ball onto the dribbler takes time and will brake the ball in case of a shot. Analysis of games from Mexico 2012 showed that successful teams do not take much use of the dribbler, especially for shooting. The fastest way to shoot is to drive directly against the ball without stopping to move on any time.

The BallGetterPlay is useful, if no opponent is near the ball. However, this is seldom the case, so a more intelligent play should be used that can conquer the ball as long as needed to change to a play with scoring chance.
One way is just breaking the ball clear by shooting it to a position without opponent bots. A better way is to combine this with a pass. A good receiver must be selected for this.

5.3 Bringing the Ball back into Game

Game analysis showed that it is more likely to score a goal directly after bringing the ball back to game. This is because on a throw in or similar situation, the AI has more time to calculate a good strategy and prepare the robots for this strategy. The whole field does not change as much as during normal game.

The ball can be brought back to game during a throw in where the ball lies near the

border of the field, or during a direct or indirect freekick which will take place on any place on the field.

The opponent bots will usually block the line between ball and goal and have also enough time to block any possible receivers, so the best way to start is to use the chip kicker. Sumatra had no plays with chip kicker last year, because the robots had no device for this. New plays must be implemented for the WCC in Eindhoven.

The challenge for a chip kick is to receive the kick. The ball will role away if it is not stopped by a robot. A robot could receive the ball straight on its dribbler, then turn. A more efficient way, however, would be to chip the ball over the receiver in a appropriate direction and let the receiver drive to it. This way, it will have the correct direction and does not need to turn.

6 Conclusion

The aim of this report was to get an overview of all plays in Sumatra and optimize the architecture behind the play/role concept. Furthermore, helpful mechanisms for play development should be created.

There is now a list of plays that were all analyzed and tested. There are no old or unknown plays or old classes that are not used. Duplicated plays or roles were removed. Some plays were modified to support a dynamic number of roles.

In the GUI of Sumatra are more visualizations that will help developers with testing their work and a recording button enables replaying certain situation for detailed analysis.

The communication between roles and skills was completely rewritten and is now more flexible and efficient. It reduces duplicate code and makes more use of the skill system.

Overall, the base of Sumatra was significantly improved. It needs, however, still more intelligent strategies and fast behaviors.

References

[1] Robocup objective, http://www.robocup.org/about-robocup/objective/, visited on 05/22/2013.

[2] RoboCup SSL autonomous referee, https://code.google.com/p/ssl-autonomous-refbox/, visited on 05/22/2013.

[3] Ryll, A. et al., Team Description for RoboCup 2013 - TIGERS Mannheim, 2013.

[4] Klostermann, D. et al., Learning play finder for complex models and complex offensive play for SSL RoboCup, 2012.

[5] Browning, B., Bruce, J., Bowling, M., and Veloso, M., STP: Skills, tactics and plays for multi-robot control in adversarial environments, 2004.

[6] König, C. et al., Artificial Intelligence - Overall Documentation, 2011.

[7] Ommer, N., Documentation of play development, http://tigers-mannheim.de/trac/wiki/KI/Module/Pandora-Plays/Playerstellung, 2013, visited on 05/23/2013.