For the purpose of the following text, capitalised sequence of words would refer to a Java type (class), named as the sequence, with the white spaces omitted.

World Data Structures:

The data structures that are used for representing the world of the robot.

LCH Colour Settings:

Description: - Information about the colour characteristics of camera output.
- Also contains two predefined static objects of itself, representing the settings for the two pitches.

Contains: - Settings about six colour classes: Yellow T, Blue T, Ball, Plate Green, Pitch Green, and Gray (walls and circle on plate).
- The settings are: Hue boundaries, Chroma boundaries, and Luma boundaries.
Hue is the actual 'colour' of the colour: green, red, purple, and so on.
Chroma is the 'vividness' of the colour: e.g. grayish, or alarming red.
Luma is the 'brightness' of the colour: white light is the brightest, white sheet of paper is less bright, yellow is bright, and blue is not.

Camera:

Description: - Information about a camera above a pitch. Also contains two predefined static objects of itself, representing the two cameras we have.

Contains: - The minimum visual crop that encloses the pitch, and colour characteristics for the camera.

Pitch:

Description: - Non-changing information about a pitch. Also contains two predefined static objects of itself, representing the two pitches.

Contains: - Physical dimensions (minimum enclosing rectangle), goal posts, camera settings for the camera above the pitch. Coordinates are in centimetres, the origin is the centre of the pitch, the X coordinate grows from left to right and the Y coordinate grows from bottom to top.

Global Info:

Description: - Information that is match-specific (changes between matches; does not change in a single match).

Contains: - Direction of attack (left / right), Alfie being yellow or blue, and the pitch that is being played on (characteristics, rather than a boolean).

Static Ball Info:

Description: - Information about the ball that is frame-specific and can be extracted from a single frame (as opposed to multiple frames).

Contains: - Physical attributes (radius), absolute position, and time-stamp.

Static Robot Info:

Description: - Information about a robot that is frame-specific and can be extracted from a single frame (as opposed to multiple frames).

Contains: - Physical attributes (width, height), absolute position and orientation, the robot being Alfie or not, ball possession, and time-stamp.

Static Info:

Description: - Basically, a structure containing two Static Robot Info objects and one Static Ball Info object. Formally, information that is frame-specific and can be extracted from a single frame (as opposed to multiple frames).

Contains: - 1 SBI, 2 SRIs - Alfie and opponent.

Static Info History:

Description: - A wrapper around a Linked List of Static Pitch Info objects.

- Can generate lists of the Static Ball Info objects, Alfie's Static Robot Info objects, and opponent's Static Robot Info objects that are contained in the list of SPIs. Convenient.

Contains: - Nothing, just extends LinkedList <StaticPitchInfo> and adds new methods.

Dynamic Ball Info:

Description: - Extends the Static Ball Info type, thus containing all of SBI's contents. Also adds information about the ball that is frame-specific, but that can only be extracted from multiple frames.

Contains: - Physical attributes (radius), absolute position, time-stamp, 'derivative' or movement information (speed, direction of travel)

Dynamic Robot Info:

Description: - Extends the Static Robot Info type, thus containing all of SRI's contents. Also adds information about a robot that is frame-specific, but that can only be extracted from multiple frames.

Contains: - Physical attributes (width, height), absolute position and orientation, the robot being Alfie or not, time-stamp, derivative' or movement information (speed, direction of travel, turning speed and direction), and whether the robot is kicking the ball or not (REMOVE ME: do this by checking if the robot had the ball and if the ball suddenly accelerated).

Dynamic Info:

Description: - Basically, contains one Dynamic Ball Info object and two Static Ball Info objects. Actually, extends the Static Info type, thus containing 1 SBI and 2 SRIs. Thanks to DBI and DRI extending SBI and SRI, there is no need of adding new fields in the class. On construction or setting, the parameters are cast down, while on getting, they are cast back up.

- Alternative approach would be to create a simple container for the DBI and two DRI, that duplicates the code in SI. This might be faster, due to less casting, but would reduce flexibility as SI objects would not have the ability to be reused as DI objects. However, we are not taking benefit from this flexibility at the time of writing this document.

Contents: One DBI and two DRI's - one for Alfie and two for its opponent.

Making Stan happy:

As we all know the most important thing about SDP is to make me (Stan) happy.

In order to do it, you must comment on the post on Facebook that promotes this section with the digit from the following sequence that should come next after the part of the sequence that was already posted. There is no real meaning in that, but to make me happy. I will post 1, you need to follow with 415926535.

If you want to make me *extremely* happy, you might also comment on the

quality of the document and whether there are poorly explained pieces.

End of World Data Structures.

Processing Pipeline:

BEWARE: *top-down* explanation; the explanation of the previous section was * bottom-up.*

The evolutionary path that the world info takes in order to reach the planning system.

Bakery: a Static Info Consumer

Description: - The Bakery bakes series of Static Info objects into Dynamic Info objects.

Main client: Overlord (see Planning Pipeline section below).

Actions: - Stores a history of Static Info objects, uses that history to estimate derivative properties of objects: speed and direction is estimated from sequence of positions.
- Passes that information to a Dynamic Info Consumer, supplied on construction of the Bakery.

Visual Cortex: an Image Consumer

Description: - The visual cortex of Alfie. Does main processing on the image.

Main client: Bakery.

Actions: - Extracts image features, like position of the ball and the robots, and orientation of the T-shapes on the top of the robots. Passes that information to a Static Info Consumer that is supplied on construction of the Visual Cortex.

Eye:

Description: - Eye of Alfie comes naturally after Eye of the Tiger and Eye of Sauron. The class captures images of the pitch.
- Measured average frame rate at 04.03.2012: on pitch 1 is 24.75 FPS +-0.5 FPS (thanks Paul), on pitch 2 is 24.87 over 4253 frames.

Main client: Visual Cortex.

Actions: - Uses the V4L4J library to grab an image from the physical camera.
- Passes the image to an Image Consumer, that is supplied on construction of the Eye.

End of Processing Pipeline.

Planning Data Structures:

The objects that are passed between the levels of planning. Vague-to-concrete explanation.

Strategy:

Description: - The overall strategy to be employed on the field.

Producer: Overlord (see Planning Pipeline section below).

Types:
1. Offensive - try to score a goal;
2. Defensive - prevent opponent from scoring a goal;
3. Take a Penalty - score a penalty;
4. Defend a Penalty - prevent the opponent from scoring a penalty; and
5. Stealth - stop on the spot and do not move.

Operation:

Description: - Similar to military operation. Describes what actions should be taken.

Producer: Field Marshal (see Planning Pipeline section below).

Types:
1. Reallocation - move to a position and face a direction;
2. Strike - kick (hopefully the ball);
3. Charge - dribble with the ball to a position and face a particular direction at the end; and
4. Overload - stop on the spot.

Path Step:

Description: - A step of a [path that aims to achieve an operation]. Describes the type of Candy Packet to give to Alfie (see Communication section below), the state in which it would be successful and the state in which it would fail.

Types:

1. Go Forward:

Act: Tell Alfie to start moving forward, possibly specifying the distance to be covered (but not speed as this messes up arc movement).

Parameters:

A position to reach, threshold distance for success, threshold angle for failure.

Succeed:

If Alfie is within the specified threshold distance from the target point.

Fail: If Alfie's facing direction is not within the specified threshold angle from the same point.

2. Go Backwards:

Act: Start moving backwards, possibly specifying the distance to be covered (but not speed as this messes up arc movement).

Parameters:

A position to reach, threshold distance for success, threshold angle for failure.

Succeed:

If Alfie is within the specified threshold distance from the target point.

Fail: If Alfie's facing direction is not within the specified threshold angle from the same point.

3. Spin Left:

Act: Tell Alfie to start spinning anti-clock-wise, possibly specifying the angle to be covered.

Parameters:

An angle for the turn, threshold delta angle for success.

Succeed:

If Alfie is within the specified threshold delta from the specified angle.

Fail:

If Alfie is turning away from the destination angle.

4. Spin Right:

Act: Tell Alfie to start spinning clock-wise, possibly specifying the angle to be covered.

Parameters:

An angle for the turn, threshold delta angle for success.

Succeed:

If Alfie is within the specified threshold delta from the specified angle.

Fail:

If Alfie is turning away from the destination angle.

5. Arc Forward Left:

Act: Tell Alfie to start moving forward in an anti-clock-wise arc, specifying the radius and angle of the arc. Thus the centre of the arc should be at the left hand side of Alfie.

Parameters:

Radius and angle of the arc and threshold distance. (Target position should be computed on construction of the Arc Forward Left object.)

Succeed:

If Alfie is within the specified threshold distance from the target position.

Fail: Easy: If Alfie is moving away from the target position.

Hard: If the target position is farther from Alfie's circular trajectory than a specified threshold. This is hard, as the vision system might not be accurate enough to allow a precise computation of the circular trajectory of Alfie. However, if implemented, this would provide Alfie with a lot quicker reactions to problems with this Path Step.

6. Arc Forward Right:

Act: Tell Alfie to start moving forward in a clock-wise arc, specifying the radius and angle of the arc. Thus the centre of the arc should be at the right hand side of Alfie.

Parameters:

Radius and angle of the arc and threshold distance. (Target position should be computed on construction of the Arc Forward Right object.)

Succeed:

If Alfie is within the specified threshold distance from the target position.

Fail: As in Arc Forward Left.

7. Arc Backwards Left:

Act: Tell Alfie to start moving backwards in a clock-wise arc, specifying the radius and angle of the arc. Thus the centre of the arc should be at the left hand side of Alfie.

Parameters:

Radius and angle of the arc and threshold distance. (Target position should be computed on construction of the Arc Backwards Left object.)

Succeed:

If Alfie is within the specified threshold distance from the target position.

Fail: As in Arc Forward Left.

8. Arc Backwards Right:

Act: Tell Alfie to start moving backwards in an anti-clock-wise arc, specifying the radius and angle of the arc. Thus the centre of the arc should be at the right hand side of Alfie.

Parameters:

Radius and angle of the arc and threshold distance. (Target position should be computed on construction of the Arc Backwards Right object.)

Succeed:

If Alfie is within the specified threshold distance from the target position.

Fail: As in Arc Forward Left.

9. Stop:

Act: Tell Alfie to stop moving.

Parameters:

None.

Succeed:

If Alfie stops moving.

Fail: If Alfie is being pushed by the other robot.

10. Kick:

Act: Tell Alfie to kick.

Parameters:

None.

Succeed:

If Alfie managed to kick the ball.

Fail: If Alfie did not kick the ball.

End of Planning Data Structures.

Planning Pipeline:

The evolutionary path that Planning Data Structures take to get from the Bakery to the Mouth (see Communication section below).

Overlord: a Dynamic Info Consumer

Description: - "If they have no bread, let them eat cake!" The popularity of this misquotation and the subjects of its metaphor suggest that the products of a bakery would be of interest to a majesty. Thus the name of the class that consumes the products of the Bakery and produces decisions on what Strategy to use is Overlord.

- It passes the Strategy to a Strategy Consumer, supplied on construction of the Overlord.

- Also passes the Dynamic Info that was received down to another Dynamic Info Consumer, supplied on construction of the Overlord. Note that the two Consumers can be the same object, but the Overlord does not need to know.

Main client: Field Marshal.

Produces: Strategy.

Responsibilities:

Producing a Strategy and monitoring if it is successful or if a problem occurs.

Policy:

Planning: - Analysing the DI (how?), the Overlord comes up with a Strategy.

- After that, it checks the success of the strategy or if a problem occurred on each DI it receives. If there is either, the Overlord comes up with a new Strategy. Otherwise, just passes the DI to its Dynamic Info Consumer.

Success: 1. Strategy is Offensive - successful if we score a goal;

2. Strategy is Defensive - successful if there is no threat of the other team scoring a goal;

3. Strategy is Take a Penalty - successful if we score a goal;

4. Strategy is Defend a Penalty - successful if Alfie prevents the opponent from scoring a penalty; and

5. Strategy is Stealth - successful if Alfie stops.

Problem Exists:

1. Strategy is Offensive - problem exists if the other robot gets the ball;

2. Strategy is Defensive - problem exists if the other team scores a goal;

3. Strategy is Take a Penalty - problem exists if Alfie misses after the shot;

4. Strategy is Defend a Penalty - problem exists if the opponent scores a goal; and

5. Strategy is Stealth - problem exists if Alfie is being moved by the other robot.

Field Marshal: a Strategy Consumer and a Dynamic Info Consumer

Description: - The highest military rank in the army. Decides what Operations should be executed.

- Passes them to a Operation Consumer, supplied on construction of the Field Marshal.

- Also passes down the Dynamic Info it was given to a Dynamic Info Consumer, supplied on construction of the Field Marshal.

Main client: Path Finder.

Produces: Operation.

Responsibilities:

Producing an Operation and monitoring if it is successful or if a problem occurs.

Policy:

Planning: - Analysing the DI (how?), the Field Marshal comes up with an Operation.

- After that, it checks the success of the operation or if a problem occurred on each DI it receives. If there is either, the Field Marshal comes up with a new Operation. Otherwise, just passes the DI to its Dynamic Info Consumer.

Success: 1. Operation is Reallocation - successful if the position is reached and if Alfie is facing the given direction;

2. Operation is Strike - successful if Alfie kicks;

3. Operation is Charge - successful if the position is reached, if Alfie is facing in the given direction, and if it still has possession of the ball; and

4. Operation is Overload - successful if Alfie stops.

Problem Exists:

1. Operation is Reallocation - problem exists if the position is invalidated (the other robot is there);

2. Operation is Strike - problem exists if Alfie does not kick;

3. Operation is Charge - problem exists if the position is invalidated (the other robot is on top), or if Alfie loses possession of the ball; and

4. Operation is Overload - problem exists if Alfie is being moved by the other robot.

Path Finder: an Operation Consumer and a Dynamic Info Consumer

Description: - Decides how to execute a particular operation. Constructs a list of Path Step objects.

Main client: None

Produces: A list of Path Step objects.

Responsibilities:

- Producing and maintaining a list of Path Step objects, according to the current Operation in execution; issuing the acts of the Path Step objects; monitoring when t he current act succeeds or fails and acting accordingly.

Policy: - When none of the Path Steps is being executed, issues the action of the next one and monitors its success or failure.

- On success, issues the action of the next Path Step.

- On failure, destroys the current list, and makes a new one. The way of issuing the act of a Path Step is implemented by invoking a method of a Mouth (see Communication section below) object, supplied on the construction of the Path Finder.

End of Planning Pipeline.

Communication:

TODO: Make section more detailed

The means of communication between the computer program and the program running on Alfie.

Candy Packet:

Description: - A packet of candies for Alfie to munch on.
- He decodes them as commands, silly him.
- For the purpose of this class a pretzel is an integer, sweets are bytes and the 'brand' of the Candy Packet is the opcode of the operation to be executed.
- Consult the old documentation for more detailed explanation.

Mouth:

Description: - Sends Candy Packet objects to the Ear (see below) through blue-tooth.

Location: On the computer.

Ear:

Description: - Receives Candy Packet objects from the Mouth through blue-tooth.

Location: On the robot.

End of Communication.

Control:

TODO: Make section more detailed

The classes that are used for controlling the robot, setting up, and feedback.

Artist:

Description: Previously Image Previewr. Draws image information in a window.

Commander Control Station:

Description: - Used for controlling the robot and setting up global data.
- Start up sequence is: Connect, Run, Use the buttons for left or right goal.
- During a match: Use the Start Planning to start the robot, and STOP to stop it.

Conversation:

Description: Used for quick test of Mouth-to-Ear communication.

End of Control.

Packages:

Description of the packages in the Commander Project.

group2.sdp.pc.breadbin:

Contains World Data Structures, that are created in the Visual Cortex and the Bakery.

group2.sdp.pc.controlstation:

Contains the Commander Control Station and the Conversation.

group2.sdp.pc.globalinfo:

Contains World Data Structures, that are used globally. Those are changed only in the Commander Control Station.

group2.sdp.pc.mouth:

Contains the Mouth and Mouth Interface.

group2.sdp.pc.planner:

Contains the classes from the Planner Pipeline.

group2.sdp.pc.planner.operation:

Contains the Operation classes.

group2.sdp.pc.planner.strategy:

Contains the Strategy enumeration.

group2.sdp.pc.vision:

Contains the Artist, the Bakery, the Eye, the Visual Cortex, and the LCH Colour classes.

group2.sdp.pc.vision.skeleton:

Contains abstract classes and interfaces for the classes in the vision package.

End of Packages.

Making Stan happy 2:

Similarly to the previous part, I am quite unhappy after spending a total of more than 18 hours this weekend on writing this documentation. So if you want to make me happy, comment on the post on Facebook, that is advertising this file with your previous digit, divided by 10^(your order of commenting), multiplied by 0.318181818. So since I would be first, I would post 1 / 10^(1) * 0.318181818 = 0.0318181818.

This documentation should be supplemented by pictures and graphs, to make it easier to understand. In its current plain-text format, despite my desperate efforts, it might not be completely obvious how things work together. Please spend a minute or two on on trying to figure out things if they look very confusing. If you do not manage to figure them out, let me know. I just wanted to make it 512 lines, so this is the purpose of the current one.