# YOrk Reinforcement Learning Library (YORLL)

***Peter Scopes, Vibhor Agarwal, Sam Devlin, Kirk Efthymiadis,***
***Kleanthis Malialis, Dorothy Thato Kentse, Daniel Kudenko***
*Reinforcement Learning Group,*
*Department of Computer Science,*
*University of York, UK*
*pds506@york.ac.uk*

## Contents

# Introduction

The YOrk Reinforcement Learning Library (YORLL) has been developed by the Reinforcement Learning (RL) Research Group at the University of York, UK. Originally YORLL was created for use by the members of the research group to be used as a common platform for experiments. It has since been decided that others could benefit from this resource and common platform.
This paper will introduce YORLL, its features, purpose, and concepts; give a high-level description of the structure and workings; and finally provide a basic introduction of how to use the library. There is also an appendix on the code style employed in the library and an appendix on how to contribute to this library.

## Purpose

When implementing experiments and algorithms, especially within the same research area, there tends to be an overlap of code from one project to the next. There is also a lot of testing that comes with the re-writing of code. Furthermore within a research group it is useful to be able to share work and ideas, and for this to be done quickly and easily. YORLL aims to curb all these problems without enforcing too many restrictions on the user.
YORLL is a modular, component based C/C++ library; it aims to provide a framework for rapid development of Reinforcement Learning (RL) experiments. It has been designed to be quick, flexible, and easy-to-use. It has been designed with the Multi-Agent paradigm in mind but can just as easily be used to run Single-Agent experiments. To our best knowledge this is the first publicly released library to offer this functionality.
The purpose of YORLL is, therefore, to be a base of code which provides the user with a means to quickly and easily create experiments, minimize the amount of code that needs to be written whilst maintaining flexibility, and be a framework so that code can be shared within research groups. In this light it is hoped that YORLL will become a new standard for RL research and continue from where other libraries, such as RL-Glue[1], have left us.

## Overview of features

The following is a basic overview of the features of YORLL:

- Runs single- and multi-agent environments
- Flexible and easy-to-use
- Extendable
- Component based design
- A number of learning algorithms already implemented (and more to come).
  - Q-Learning
  - SARSA
  - Argmax
  - -Greedy
  - SoftMax
- Configuration files for batch running without recompilation

## Basic Structure

YORLL is set up so that there is a minimum amount of code needed to be written for each experiment without losing flexibility in what can be created nor running speed of each project. The library consists of two overarching parts: the library resources and experiments. The library resources are made up of four subsections: agents, environments, extensions, and utilities. The agent subsection consists of learning algorithm interface and algorithms, action selection algorithms, and the interface for agents. The environment subsection contains the Environment,

[1] RL-Glue can be found at http://groups.google.com/group/rl-glue, at the time of writing it is no longer under active development and was never intended for multi-agent experiments.

State, and State Representation interfaces; State and State Representation differ in that the State is the complete state of the environment for a given agent and the State Representation is processed version of the State an agent uses to learn, these can be the same thing. The extensions subsection the place to put non-essential pieces of code that have been made such that they can be used in more than one experiment, e.g. Tile Coding. Finally the utility subsection holds all the utility objects such as the File Reader, Configuration File Reader, and other data structures that aren't specific to RL, e.g. Splaying Binary Tree.

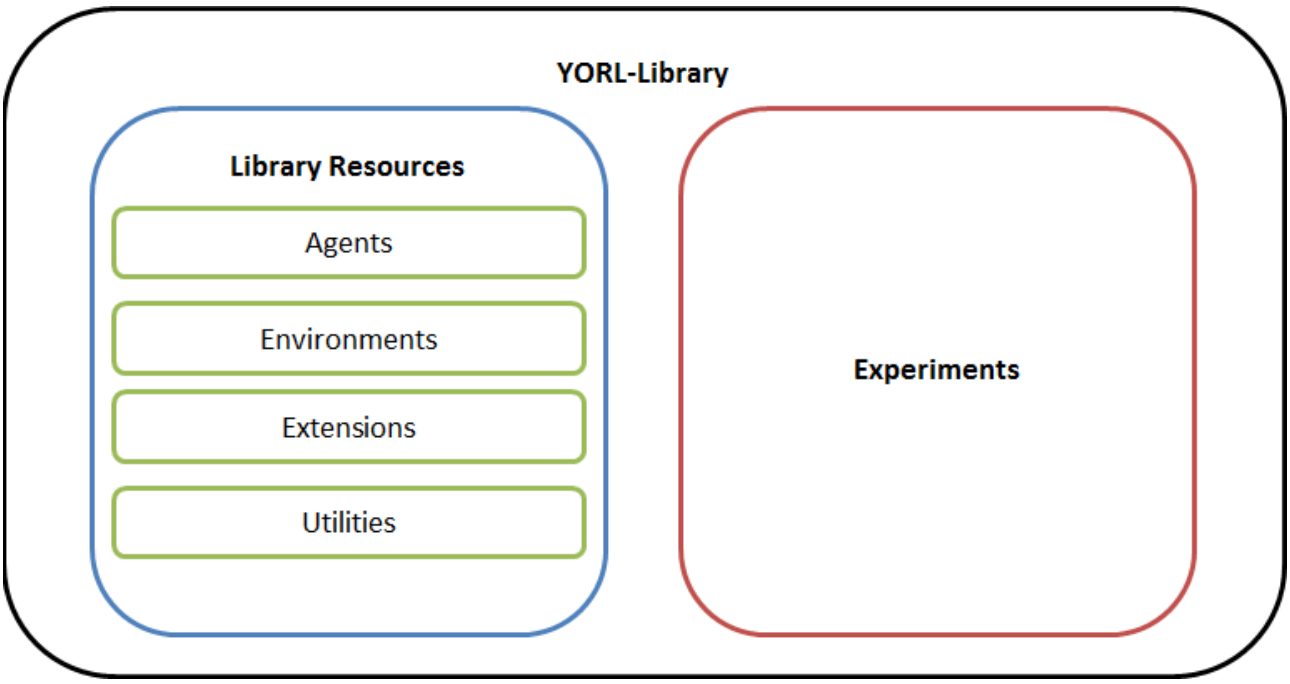


**Figure 1:** The basic structure of the YORL– Library

Implementations of experiments should contain all experiment specific aspects such as the environment, state, and agent(s) classes.

# YORLL Concepts

## Logical Loop

*Outline*

An experiment begins by loading the configuration file, then it repeatedly executes runs, and terminates. In an episodic environment a run consists of repeated episodes where an episode is a sequence of time steps being performed until a maximum number of steps is reached or the environment enters a terminal state. In a continuous environment a run is a sequence of time steps being performed until a terminal state is reached[2].

A configuration file is either specified at the command line or a default file of the experiment is used; the configuration file allows the user to run different parameter settings, repeats of the experiments, etc. without re-compiling the project.

Before the runs are executed the environment and agent(s) are instantised[3] then at the beginning of each run they are (re)initialised so that each run begins as if no other runs have happened. At the

[2] YORLL could be modified to allow for non-discrete time but it was deemed unnecessary to build in the extra complications since it is rarely needed

[3] In the case of experiments with a dynamic number of agents the agents can be created at any time since the experiment file is written on a per experiment basis.

beginning of each episode the environment and agent(s) are reset such that they are ready to begin the episode once again.

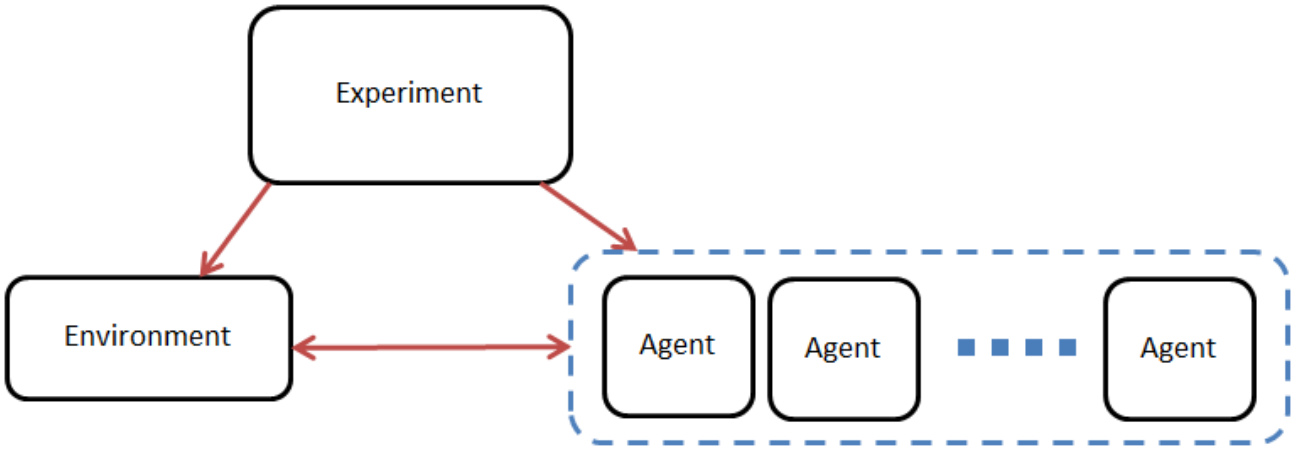


**Figure 2:** Objects within the system (arrows indicate function calls)

As Figure 2 depicts the experiment can call functions from both the environment and agent(s), an agent can all call functions from the environment and vice versa. In this way the environment can request from the agent(s) its action(s) for the current state and the agent can perceive and act upon the environment.

*Within a single step*

Since YORLL was designed for both single- and multi-agent environments the step cycle is more complex than a library for just single-agent environments[4]. Firstly each agent must have already perceived the current state, initially this is done when each agent is reset at the beginning of each episode, but then at the end of the step when they are informed of their reward. The environment calls each agent to decide upon its next action to perform; the ordering of this can be one of two ways: Randomly or in a defined order, this decision is given to the user of the library. Once each agent has notified the environment of the action it wishes to perform the environment must perform conflict resolution since actions may be mutually exclusive, how conflict resolution is handled is left in the hands of the user but the three possible options are:

- *Mutual* - decline all agents involved in the conflict
- *Ordered* - decline agents in a specified order, such as priority or role
- *Random* - decline agents in a random order

Finally each agent must be informed of the reward it has received and then if it has reached a terminal state the reward for reaching said terminal state.

*Nuances*

As with any system there are some nuances within the system which have come from design decisions. This section aims to list what these are and, if needed, describe why they exist.

- Actions are requested then confirmed
    - This is a result of allowing for multi-agent environments since the environment may need to resolve conflicts that arise from mutually exclusive actions of one or more agents.
    - This is seen during the step process where an agent doesn't get an immediate reward after informing the environment of its actions but must wait for the

[4] When creating a single-agent environment the step cycle can be made simpler if really desired but it is recommended to keep a common structure among experiments.

environment to inform it of the reward.

- Agents must perceive on reset
    - This is the result of most RL algorithms the new state that the agent has arrived in is needed to learn.
    - Since the agent needs to perceive the environment after the action has been confirmed (when the reward is being given) it was decided to perceive during the reset action instead of perceiving twice per step.

## Conclusion

This paper has introduced YOrk Reinforcement Learning Library (YORLL) and outlined its purpose, concepts, and structure. To the best of our knowledge YORLL is the first publicly available library that supports multi-agent environments. It is our hope that YORLL will become a standard for implementations of Reinforcement Learning (RL) experiments to be built. Please see the appendices for more information on how to use the library, code styles employed, and how to submit extensions.

# Appendices

## Appendix A - How to Use

*Installation*

YORLL was developed in Visual Studios 2010 Professional™ as a console 32 application. This installation guide will show how to set up YORLL in Visual Studios:

1. Download the zipped library and extracted into your project folder.
2. Open Visual Studios and select “File > New > Project From Existing Code...”
3. Follow the wizard ensuring you choose “console 32 application”
4. Once the project has been created select “Project > Properties”
5. Next select “VC++ Directories” and add “$(ProjectDir)\YORLL\headers” to “Include Directories”
6. In “Solution Explorer” select the “Show All Files” and in both headers and src exclude all experiments apart from the one you wish to run
7. Finally select “Build > Build Solution”

*Creating an Experiment*

Please note that before creating your own experiment it is recommended that you first inspect how how the example experiments work; the most simple example SingleAgentMaze.

To create your own experiment first you must create a new folder with the name of the experiment in the “Experiments” folder in both “headers” and “src”. There are five files that are needed for an experiment: the experiment which has the main function, implementations of an agent, the environment, and the state, and finally the configuration file; you should use the example experiments to learn how exactly to do this.

## Appendix B - Code styling

In general YORLL follows the rules for code styling given by *C++ Programming Style Guidelines* which can be found at http://geosoft.no/development/cppstyle.html and the *Doxygen* rules for commenting which can be found at http://www.stack.nl/~dimitri/doxygen/docblocks.html.

## Appendix C - Submitting extensions

We are very welcome to receiving extensions that you may have created and wish to share. Before doing so please ensure you have read the section of this paper about “Coding Styling”. Please note any extensions we add to the library will have to be written in an easily reusable way. We shall also test the code ourselves and may alter the interface so that it fits better into the library.
Please email pds506@york.ac.uk with any submissions.