MC-AIXI-CTW

1

Generated by Doxygen 1.7.1

Sat Apr 23 2011 11:53:08

Contents

Chapter 1

MC-AIXI-CTW

The Monte-Carlo AIXI agent (Veness et al.) is an approximation of the universally optimal AIXI reinforcment learning agent. The purpose of this software package is to provide a simple implementation of the basic MC-AIXI-CTW agent. For a more heavy-weight implementation, see Joel Veness' code from which this simpler version is derived. Similarly, the current documentation and associated tutorial gives a very brief and mostly non-technical overview of the MC-AIXI-CTW agent. For a thorough and formal treatment we recommend the original paper.

The MC-AIXI-CTW agent seeks to interact intelligently within a particular environment. The environment can be just about anything, from a chess game to the entire universe. The agent interacts with the environment by performing actions (such as moving a piece on a chess board) and the environment interacts with the agent by providing observations (such as an image from a camera) and rewards for the agents actions. It is the goal of the agent to use its past interaction history to learn how to choose the actions which will lead to the greatest long-term reward.

There are two main components which combine to give the agent's action selection policy. The first is a model of the environment with which the agent attempts to model how the environment works and hence to predict how likely any given outcome is. Second is an algorithm for estimating the expected reward of each possible action by using the probabilities associated with the environment model. In particular, the MC-AIXI-CTW agent uses context-tree weighting for the environment model and the ρUCT search algorithm for estimating the expected rewards.

The software is divided into several components, each of which corresponds roughly with some component of the agent.

- The main.cpp file contains the code which sets up and runs the agent/environment interaction loop.
- High-level control of the agent is controlled by the Agent class with the the details of the environment model and action selection handled other classes as follows:
  - The agent's environment model is implemented by the ContextTree and CTNode classes. The class ModelUndo is used to store the information required in order to return the environment model to a previous state.
  - The ρUCT search algorithm is implemented in part by the SearchNode class and in part by the Agent class.
- Each environment inherits from the Environment class. The currently implemented environments include:
  - CoinFlip
  - ExtendedTiger

  - KuhnPoker
  - Maze
  - PacMan
  - RockPaperScissors
  - TicTacToe
  - Tiger
- Miscellaneous utility functions are contained in util.cpp and util.hpp.

For more details on the code including how to run, configure, log, or make changes please see the accompanying tutorial (in the tutorial subfolder of the MC-AIXI-CTW package).

Chapter 2

Data Structure Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 Agent Class Reference

`#include <agent.hpp>`

Collaboration diagram for Agent:

Public Member Functions

- age_t age () const
- Agent (options_t &options, Environment const &env)
- reward_t averageReward () const
- action_t genAction () const
- void genPercept (percept_t &observation, percept_t &reward)
- void genPerceptAndUpdate (percept_t &observation, percept_t &reward)
- action_t genRandomAction () const
- double getPredictedActionProb (action_t action)
- int historySize () const
- int horizon () const
- update_t lastUpdate (void) const
- int maxAction () const
- int maxBitsNeeded () const
- double maxReward () const
- void modelRevert (const ModelUndo &mu)
- int modelSize () const
- void modelUpdate (percept_t observation, percept_t reward)
- void modelUpdate (action_t action)
- double perceptProbability (percept_t observation, percept_t reward) const
- reward_t playout (int horizon)
- void reset (void)
- action_t search (void)
- reward_t totalReward () const
- ~Agent ()

Private Member Functions

- action_t decodeAction (const symbol_list_t &symlist) const
- percept_t decodeObservation (const symbol_list_t &symlist) const
- void decodePercept (const symbol_list_t &symlist, percept_t &observation, percept_t &reward)
- percept_t decodeReward (const symbol_list_t &symlist) const
- void encodeAction (symbol_list_t &symlist, action_t action) const
- void encodePercept (symbol_list_t &symlist, percept_t observation, percept_t reward) const

Private Attributes

- ContextTree * m_ct
- Environment const & m_env
- int m_horizon
- update_t m_last_update
- int m_learning_period
- int m_mc_simulations
- options_t & m_options
- SearchNode * m_search_tree
- age_t m_time_cycle
- reward_t m_total_reward

5.1.1 Detailed Description

The Agent class represents a MC-AIXI-CTW agent. It includes much of the high-level logic for choosing suitable actions. In particular, the agent maintains an internal model of the environment using a context tree Agent::m_ct. It uses this internal model to to predict the probability of future outcomes:

- Agent::getPredictedActionProb()
- Agent::perceptProbability()

as well as to generate actions and percepts according to the model distribution:

- Agent::genAction()
- Agent::genPercept()
- Agent::genPerceptAndUpdate()
- Agent::genRandomAction()

Actions are chosen via the UCT algorithm, which is orchestrated by a high-level search function and a playout policy:

- Agent::search()
- Agent::playout()
- Agent::m_horizon
- Agent::m_mc_simulations
- Agent::m_search_tree

Several functions decode/encode actions and percepts between the corresponding types (i.e. action_t, percept_t) and generic representation by symbol lists:

- Agent::decodeAction()
- Agent::decodeObservation()
- Agent::decodePercept()
- Agent::decodeReward()
- Agent::encodeAction()
- Agent::encodePercept()

There are various attributes which describe the agent and it's interaction with the environment so far:

- Agent::age()
- Agent::averageReward()
- Agent::historySize()
- Agent::horizon()
- Agent::lastUpdate()
- Agent::maxAction()
- Agent::maxBitsNeeded()
- Agent::maxReward()
- Agent::totalReward()

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Agent::Agent (options_t & *options*, Environment const & *env*)

Construct a learning agent from the configuration arguments and environmet.

Parameters

options The configuration options.

env The environment the agent will interact with.

5.1.2.2 Agent::~Agent (void)

Destry the agent and the corresponding context tree.

5.1.3 Member Function Documentation

5.1.3.1 age_t Agent::age (void) const

Current age of the agent in cycles.

5.1.3.2 reward_t Agent::averageReward (void) const

The average reward received by the agent at each time step.

5.1.3.3 action_t Agent::decodeAction (const symbol_list_t & *symlist*) const [private]

Decode an action from the beginning of a list of symbols.

Parameters

symlist The symbol list to decode the action from.

Returns

The decoded action.

5.1.3.4 percept_t Agent::decodeObservation (const symbol_list_t & *symlist*) const [private]

Decode an observation from the beginning of a list of symbols.

Parameters

symlist The symbol list to decode the observation from.

Returns

The decoded observation.

5.1.3.5 void Agent::decodePercept (const symbol_list_t & *symlist,* percept_t & *observation,* percept_t & *reward*) [private]

Decode a percept (observation and reward) from the beginning of a list of symbols.

Parameters

symlist The symbol list to decode the observation from.

observation Receives the decoded observation.

reward Receives the decoded reward.

5.1.3.6 percept_t Agent::decodeReward (const symbol_list_t & *symlist*) const [private]

Decode a reward from the beginning of a list of symbols.

Parameters

symlist The symbol list to decode the reward from.

Returns

The decoded reward.

5.1.3.7 void Agent::encodeAction (symbol_list_t & *symlist,* action_t *action*) const [private]

Encode an action as a list of symbols.

Parameters

symlist The symbol list to encode the action to.

action The action to encode.

5.1.3.8 void Agent::encodePercept (symbol_list_t & *symlist,* percept_t *observation,* percept_t *reward*) const [private]

Encode a percept as a list of symbols.

Parameters

symlist The symbol list to encode the percept to.

observation The observation part of the percept to encode.

reward The reward part of the percept to encode.

5.1.3.9 action_t Agent::genAction (void) const

Generate an action distributed according to the agent's history statistics by doing rejection sampling from the context tree.

Returns

The generated action.

5.1.3.10 void Agent::genPercept (percept_t & *observation,* percept_t & *reward*)

Generate a percept distributed according to the agent's history statistics by sampling from the context tree.

Parameters

observation Receives the observation part of the generated percept.

reward Receives the reward part of the generated percept.

5.1.3.11 void Agent::genPerceptAndUpdate (percept_t & *observation,* percept_t & *reward*)

Generate a percept distributed according to the agent's history statistics, and update the context tree with it.

Parameters

observation Receives the observation part of the generated percept.

reward Receives the reward part of the generated percept.

5.1.3.12 action_t Agent::genRandomAction (void) const

Generate an action uniformly at random.

Returns

The generated action.

5.1.3.13 double Agent::getPredictedActionProb (action_t *action*)

Probability of selecting an action according to the agent's internal model of it's own behaviour.

Parameters

action The action we wish to find the likelihood of.

Returns

The probability of the agent selecting action.

5.1.3.14 int Agent::historySize (void) const

The length of the stored history for an agent.

5.1.3.15 int Agent::horizon (void) const

The length of the search horizon used by the agent.

5.1.3.16 update_t Agent::lastUpdate (void) const [inline]

True if the last update was a percept, false if it was an action.

5.1.3.17 int Agent::maxAction ( ) const [inline]

The "maximum" action the agent can execute.

5.1.3.18 int Agent::maxBitsNeeded ( ) const

The maximum number of bits to encode either an action or percept.

5.1.3.19 double Agent::maxReward ( ) const [inline]

The maximum possible reward the agent can receive in a single cycle.

5.1.3.20 void Agent::modelRevert (const ModelUndo & *mu*)

Revert the agent's model of the world to that of a previous time cycle.

5.1.3.21 int Agent::modelSize ( ) const

5.1.3.22 void Agent::modelUpdate (action_t *action*)

Update the agent's model of the world after performing an action.

Parameters

action The action that the agent performed.

5.1.3.23 void Agent::modelUpdate (percept_t *observation,* percept_t *reward*)

Update the agent's model of the world with a percept from the environment

Parameters

observation The observation that was received.

reward The reward that was received.

5.1.3.24 double Agent::perceptProbability (percept_t *observation,* percept_t *reward*) const

Probability of receiving a particular percept (observation and reward) according to the agent's environment model.

Parameters

observation The observation part of the percept we wish to find the likelihood of.

reward The reward part of the percept we wish to find the likelihood of.

Returns

The probability of observing the (observation, reward) pair.

5.1.3.25 reward_t Agent::playout (int *horizon*)

Simulate agent/enviroment interaction for a specified amount of steps where agent actions are chosen uniformly at random and percepts are generated from the agents environment model.

Parameters

agent The agent doing the sampling.

playout_len The number of complete action/percept steps to simulate.

Returns

The total reward from the simulation.

5.1.3.26 void Agent::reset (void)

Resets the agent and clears the context tree.

5.1.3.27 action_t Agent::search (void)

Determine the best action for the agent using Monte-Carlo Tree Search (predictive UCT).

Returns

The best action as determined by the sampling.

5.1.3.28 reward_t Agent::totalReward (void) const

The total accumulated reward across an agents lifespan.

5.1.4 Field Documentation

5.1.4.1 ContextTree* Agent::m_ct [private]

Context tree representing the agent's model of the environment.

5.1.4.2 Environment const& Agent::m_env [private]

A reference to the environment the agent interacts with.

5.1.4.3 int Agent::m_horizon [private]

The length of the agent's planning horizon.

5.1.4.4 update_t Agent::m_last_update [private]

The type of the last update (action or percept).

5.1.4.5 int Agent::m_learning_period [private]

The number of cycles during which the agent learns.

5.1.4.6 int Agent::m_mc_simulations [private]

The number of simulations to conduct when choosing new actions via the UCT algorithm.

5.1.4.7 options_t& Agent::m_options [private]

Stores the configuration options.

5.1.4.8 SearchNode* Agent::m_search_tree [private]

The root node of the UCT search tree.

5.1.4.9 age_t Agent::m_time_cycle [private]

The number of interaction cycles the agent has been alive.

5.1.4.10 reward_t Agent::m_total_reward [private]

The total reward received by the agent.

The documentation for this class was generated from the following files:

- agent.hpp
- agent.cpp

5.2 CoinFlip Class Reference

`#include <coinflip.hpp>`

Inheritance diagram for CoinFlip:

Collaboration diagram for CoinFlip:

Public Member Functions

- CoinFlip (options_t &options)
- virtual action_t maxAction () const
- virtual percept_t maxObservation () const
- virtual percept_t maxReward () const
- virtual void performAction (const action_t action)
- virtual std::string print () const

Private Attributes

- double m_probability

Static Private Attributes

- static const action_t aHeads = 1
- static const action_t aTails = 0
- static const double cDefaultProbability = 0.7
- static const percept_t oHeads = 1
- static const percept_t oTails = 0
- static const percept_t rLoss = 0
- static const percept_t rWin = 1

5.2.1 Detailed Description

A biased coin is flipped and the agent is tasked with predicting how it will land. The agent receives a reward of CoinFlip::rWin for a correct prediction and CoinFlip::rLoss for an incorrect prediction. The observation specifies which side the coin landed on (CoinFlip::oTails or CoinFlip::oHeads). The action corresponds to the agent's prediction for the next coin flip (CoinFlip::aTails or CoinFlip::aHeads).

Domain characteristics:

- environment: "coin-flip"
- maximum action: 1 (1 bit)
- maximum observation: 1 (1 bit)
- maximum reward: 1 (1 bit)

Configuration options:

- coin-flip-p (optional): the probability the coin lands on heads (CoinFlip::oHeads). Must be a floating point number between 0.0 and 1.0 inclusive. Default value is CoinFlip::cDefaultProbability.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 CoinFlip::CoinFlip (options_t & *options*)

5.2.3 Member Function Documentation

5.2.3.1 virtual action_t CoinFlip::maxAction ( ) const [inline, virtual]

The maximum possible action.

Implements Environment.

5.2.3.2 virtual percept_t CoinFlip::maxObservation ( ) const [inline, virtual]

The maximum possible observation.

Implements Environment.

5.2.3.3 virtual percept_t CoinFlip::maxReward ( ) const [inline, virtual]

The maximum possible reward.

Implements Environment.

5.2.3.4 void CoinFlip::performAction (const action_t *action*) [virtual]

Receives the agent's action and calculates the new environment percept.

Implements Environment.

5.2.3.5 std::string CoinFlip::print (void) const [virtual]

Reimplemented from Environment.

5.2.4 Field Documentation

5.2.4.1 const action_t CoinFlip::aHeads = 1 [static, private]

ACtion: agent predicts the coin will land on heads.

5.2.4.2 const action_t CoinFlip::aTails = 0 [static, private]

Action: agent predicts the coin will land on tails.

5.2.4.3 const double CoinFlip::cDefaultProbability = 0.7 [static, private]

Default value of CoinFlip::m_probability.

5.2.4.4 double CoinFlip::m_probability [private]

Probability of throwing heads (CoinFlip::oHeads). Default value is CoinFlip::cDefaultProbability.

5.2.4.5 const percept_t CoinFlip::oHeads = 1 [static, private]

Observation: the coin lands on heads.

5.2.4.6 const percept_t CoinFlip::oTails = 0 [static, private]

Observation: the coin lands on tails.

5.2.4.7 const percept_t CoinFlip::rLoss = 0 [static, private]

Reward: agent incorrectly predicted the toss.

5.2.4.8 const percept_t CoinFlip::rWin = 1 [static, private]

Reward: agent correctly predicted the toss.

The documentation for this class was generated from the following files:

- coinflip.hpp
- coinflip.cpp

5.3 ContextTree Class Reference

`#include <predict.hpp>`

Collaboration diagram for ContextTree:

Public Member Functions

- void clear (void)
- ContextTree (const int depth)
- size_t depth (void) const
- void genRandomSymbols (symbol_list_t &symbols, const int bits)
- void genRandomSymbolsAndUpdate (symbol_list_t &symbols, const int bits)
- size_t historySize (void) const
- double logBlockProbability (void) const
- weight_t predict (const symbol_t symbol)
- weight_t predict (symbol_list_t const &symbols)
- void revert (const int num_symbols)
- void revert (void)
- void revertHistory (const int num_symbols)
- size_t size (void) const
- void update (symbol_list_t const &symbols)
- void update (const symbol_t symbol)
- void updateHistory (const symbol_t symbol)
- void updateHistory (symbol_list_t const &symbols)
- ~ContextTree (void)

Private Member Functions

- void updateContext (void)

Private Attributes

- CTNode ** m_context
- int m_depth
- symbol_list_t m_history
- CTNode * m_root

5.3.1 Detailed Description

The high-level interface to an action-conditional context tree. Most of the mathematical details are implemented in the CTNode class, which is used to represent the nodes of the tree. ContextTree stores a reference to the root node of the tree (ContextTree::m_root), the history of updates to the tree (ContextTree::m_history), and the maximum depth of the tree (ContextTree::m_depth). It is primarily concerned with calling the appropriate functions in the appropriate nodes in order to deliver certain functionality:

- Updating the context tree and reverting previously made updates.
  - ContextTree::update(symbol_t) and ContextTree::update(const symbol_list_t&) update the tree and the history after the agent has observed new percepts.
  - ContextTree::updateHistory(symbol_t) and ContextTree::updateHistory(const symbol_list_t&) update just the history after the agent has executed an action.
  - ContextTree::revert() undoes the last update to the tree.
  - ContextTree::revertHistory() deletes the recent history.
- Predicting the probability of future outcomes (ContextTree::predict()).

- Sampling sequences of symbols from the context tree statistics.
  - ContextTree::genRandomSymbolAndUpdate() samples a sequence from the context tree, updating the tree with each bit as it is sampled.
  - ContextTree::genRandomSymbols() samples a sequence of a specified length, updating the tree with each bit as it is sampled, then reverting all the updates so that the tree is in the same state as it was before the sampling.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ContextTree::ContextTree (const int *depth*)

Create a context tree of specified maximum depth. Only allocates memory for the root node, other nodes are created lazily as needed.

Parameters

depth The maximum depth of the context tree.

5.3.2.2 ContextTree::~ContextTree (void)

Destroy the context tree and all the nodes referenced by the tree.

5.3.3 Member Function Documentation

5.3.3.1 void ContextTree::clear (void)

Clears the entire context tree including all nodes and history.

5.3.3.2 size_t ContextTree::depth (void) const [inline]

Returns

The maximum depth of the context tree.

5.3.3.3 void ContextTree::genRandomSymbols (symbol_list_t & *symbols*, const int *bits*)

Generate a bit string of a specified length by sampling from the context tree.

Parameters

symbols Stores the generated string.

bits The number of bits to generate.

5.3.3.4 void ContextTree::genRandomSymbolsAndUpdate (symbol_list_t & *symbols*, const int *bits*)

Generate a bit string of a specified length by sampling from the context tree and update the tree with the generated bits.

Parameters

symbols Stores the generated string.

bits The number of bits to generate.

5.3.3.5 size_t ContextTree::historySize (void) const [inline]

Returns

The size of the stored history.

5.3.3.6 double ContextTree::logBlockProbability (void) const

The logarithm of the block probability of the history sequence.

5.3.3.7 weight_t ContextTree::predict (const symbol_t *symbol*)

The estimated probability of observing a particular symbol. Given a history sequence h and a symbol y, the estimated probability is given by

$$\rho(y|h) = \frac{\rho(hy)}{\rho(h)}$$

where $\rho(h) = P_w^\epsilon(h)$ is the weighted probability estimate of observing h evaluated at the root node ϵ of the context tree.

Parameters

symbol The symbol to estimate the conditional probability of. A false value corresponds to $\rho(0|h)$ and a true value to $\rho(1|h)$.

5.3.3.8 weight_t ContextTree::predict (symbol_list_t const & *symbols*)

The estimated probability of observing a particular sequence of symbols. Given a history sequence h and a sequence of symbols y, the estimated probability $\rho(y|h)$ of observing y is

$$\rho(y|h) = \frac{\rho(hy)}{\rho(h)}$$

where $\rho(h) = P_w^\epsilon(h)$ is the weighted probability estimate of observing h evaluated at the root node ϵ of the context tree.

Parameters

symbols The sequence of symbols to estimate the conditional probability of.

5.3.3.9 void ContextTree::revert (void)

Restores the context tree to as it was immediately prior to the previous update (CTNode::update()).

5.3.3.10 void ContextTree::revert (const int *num_symbols*)

Restores the context tree to its state prior to a specified number of updates

Parameters

num_symbols The number of updates (symbols) to revert.

5.3.3.11 void ContextTree::revertHistory (const int *num_symbols*)

Shrinks the history down to a former size without changing the context tree.

5.3.3.12 size_t ContextTree::size (void) const `[inline]`

Returns

number of nodes in the context tree.

5.3.3.13 void ContextTree::update (symbol_list_t const & *symbols*)

Update the context tree with a list of symbols. Equivalent to calling ContextTree::update() once for each symbol.

Parameters

symbols The symbols with which to update the tree. The context tree is updated with symbols in the order they appear in the list.

5.3.3.14 void ContextTree::update (const symbol_t *symbol*)

Update the context tree with a new binary symbol. Recalculate the log weighted probabilities and log KT estimates for each affected node.

Parameters

symbol The symbol with which to update the tree.

5.3.3.15 void ContextTree::updateContext (void) `[private]`

Calculates which nodes in the context tree correspond to the current context and adds them to CTNode::m_context in order from root to leaf. In particular, ContextTree::m_context[0] will always correspond to the roo node and ContextTree::m_context [m_depth] corresponds to the relevant leaf node. Creates the nodes if they do not exist.

5.3.3.16 void ContextTree::updateHistory (symbol_list_t const & *symbols*)

Append symbols to the history without updating the context tree.

Parameters

symbols The list of symbols to add to the history.

5.3.3.17 void ContextTree::updateHistory (const symbol_t *symbol*)

Append a symbol to the history without updating the context tree.

Parameters

symbol The symbol to add to the history.

5.3.4 Field Documentation

5.3.4.1 CTNode** ContextTree::m_context [private]

An array of length CTNode::m_depth + 1 used to hold the nodes in the context tree that correspond to the current context. It is important to ensure that ContextTree::updateContext() is called before accessing the contents of this array as they may otherwise be inaccurate.

5.3.4.2 int ContextTree::m_depth [private]

The maximum depth of the context tree.

5.3.4.3 symbol_list_t ContextTree::m_history [private]

The agent's history.

5.3.4.4 CTNode* ContextTree::m_root [private]

The root node of the context tree.

The documentation for this class was generated from the following files:

- predict.hpp
- predict.cpp

5.4 CTNode Class Reference

`#include <predict.hpp>`

Collaboration diagram for CTNode:

Public Member Functions

- const CTNode * child (const symbol_t sym) const
- bool isLeafNode (void) const
- weight_t logKT (void) const
- weight_t logProbability (void) const
- int size (void) const
- int visits (void) const

Private Member Functions

- CTNode (void)
- weight_t logKTMultiplier (const symbol_t symbol) const
- void revert (const symbol_t symbol)
- void update (const symbol_t symbol)
- void updateLogProbability (void)
- ~CTNode (void)

Private Attributes

- CTNode * m_child [2]
- int m_count [2]
- weight_t m_log_kt
- weight_t m_log_probability

Friends

- class ContextTree

5.4.1 Detailed Description

The CTNode class represents a node in an action-conditional context tree. The purpose of each node is to calculate the weighted probability of observing a particular bit sequence. In particular, denote by n the current node, by $n0$ and $n1$ the child nodes, by h_n the subsequence of the history relevant to node n, and by a and b the number of zeros and ones in h_n. Then the weighted block probability of observing h_n at node n is given by

$$P_w^n(h_n) := \begin{cases} \Pr_{\text{kt}}(h_n) & \text{if } n \text{ is a leaf node} \\ \frac{1}{2}\Pr_{\text{kt}}(h_n) + \frac{1}{2}P_w^{n0}(h_{n0})P_w^{n1}(h_{n1}) & \text{otherwise} \end{cases}$$

where $\Pr_{\text{kt}}(h_n) = \Pr_{\text{kt}}(a, b)$ is the Krichevsky-Trofimov (KT) estimator defined by the relations

$$\Pr_{\text{kt}}(a+1, b) = \frac{a + 1/2}{a + b + 1}\Pr_{\text{kt}}(a, b)$$

$$\Pr_{\text{kt}}(a, b+1) = \frac{b + 1/2}{a + b + 1}\Pr_{\text{kt}}(a, b)$$

and the base case $\Pr_{\text{kt}}(0, 0) := 1$. In both relations, the fraction is referred to as the update multiplier and corresponds to the probability of observing a zero (first relation) or a one (second relation) given we have seen a zeros and b ones.

Due to numerical issues, the implementation uses logarithmic probabilities $\ln P_w^n(h_n)$ and $\ln \Pr_{\text{kt}}(h_n)$ rather than normal probabilities. These probabilities are recalculated during updates (CTNode::update()) and reversions (CTNode::revert()) to the context tree that involve the node.

- The KT estimate is stored using CTNode::m_log_kt and accessed through CTNode::logKT(). It is updated from the previous estimate by multiplying with the update multiplier as calculated by CTNode::logKTMultiplier().
- The weighted probability is stored using CTNode::m_log_probability and accessed through CTNode::logProbability(). It is recalculated by CTNode::updateLogProbability().

In order to calculate these probabilities, CTNode also stores:

- Links to child nodes: CTNode::child(), CTNode::m_child.
- The number of zeros and ones in the history subsequence relevant to the node: CTNode::m_count.

The CTNode class is tightly coupled with the ContextTree class. Briefly, the ContextTree class

- Creates and deletes nodes.
- Tells the appropriate nodes to update/revert their probability estimates.
- Samples actions and percepts from the probability distribution specified by the nodes.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 CTNode::CTNode (void) [private]

Initialise the node.

5.4.2.2 CTNode::~CTNode (void) [private]

Destroy the node and all children.

5.4.3 Member Function Documentation

5.4.3.1 const CTNode* CTNode::child (const symbol_t *sym*) const [inline]

The child node corresponding to a particular symbol.

5.4.3.2 bool CTNode::isLeafNode (void) const [inline]

Checks if this is a leaf node.

Returns

True if the node is a leaf node, false otherwise.

5.4.3.3 weight_t CTNode::logKT (void) const [inline]

Retrieves the cached KT estimate of the log probability of the history subsequence relevant to this node. The value is computed only when the node is changed (by CTNode::update() or CTNode::revert()) and is cached in the variable CTNode:m_log_kt.

Returns

The log KT estimate $\ln \Pr_{kt}(h_{T,n}) = \ln \Pr_{kt}(0^a 1^b) = \ln \Pr_{kt}(a, b)$ where a and b denote the number of zeros and ones in the history subsequence $h_{T,n}$ relevant to this node n.

5.4.3.4 weight_t CTNode::logKTMultiplier (const symbol_t *symbol*) const [private]

Compute the logarithm of the KT-estimator update multiplier. The log KT estimate of the conditional probability of observing a zero given we have observed a zeros and b ones at the current node is

$$\ln \Pr_{\text{kt}}(0 \mid 0^a 1^b) = \ln \frac{a + 1/2}{a + b + 1}.$$

Similarly, the estimate of the conditional probability of observing a one is

$$\ln \Pr_{\text{kt}}(1 \mid 0^a 1^b) = \ln \frac{b + 1/2}{a + b + 1}.$$

Parameters

symbol The symbol for which to calculate the log KT estimate of conditional probability. False corresponds to calculating $\ln \Pr_{\text{kt}}(0 \mid 0^a 1^b)$ and true corresponds to calculating $\ln \Pr_{\text{kt}}(1 \mid 0^a 1^b)$.

Returns

The log KT estimate of the conditional probability (update multiplier).

5.4.3.5 weight_t CTNode::logProbability (void) const [inline]

Retrieves the cached weighted log probability of the history subsequence relevant to this node. The value is computed only when the node is changed (by CTNode::update() or CTNode::revert()) and is cached in the variable CTNode::m_log_probability.

Returns

The log weighted probability $\ln P_w^n$.

5.4.3.6 void CTNode::revert (const symbol_t *symbol*) [private]

Return the node to its state immediately prior to the last update. This involves updating the symbol counts, recalculating the cached probabilities, and deleting unnecessary child nodes.

Parameters

symbol The symbol used in the previous update.

5.4.3.7 int CTNode::size (void) const

The number of nodes in the tree rooted at this node.

5.4.3.8 void CTNode::update (const symbol_t *symbol*) [private]

Update the node after having observed a new symbol. This involves updating the symbol counts and recalculating the cached probabilities.

Parameters

The symbol that was observed.

5.4.3.9 void CTNode::updateLogProbability (void) [private]

Calculates the logarithm of the weighted block probability

$$\ln P_w^n := \begin{cases} \ln \Pr_{\text{KT}}(h_n) & \text{if } n \text{ is a leaf node} \\ \ln\left(\frac{1}{2}\Pr_{\text{KT}}(h_n) + \frac{1}{2}P_w^{n^0} \times P_w^{n^1}\right) & \text{otherwise} \end{cases}$$

and stores the value in CTNode::m_log_probability.

Because of numerical issues, the implementation works directly with the log probabilities $\ln \Pr_{\text{KT}}(h_n)$, $\ln P_w^{n^0}$, and $\ln P_w^{n^1}$ rather than the normal probabilities. To compute the second case of the weighted probability, we use the identity

$$\ln(a+b) = \ln a + \ln(1 + \exp(\ln b - \ln a)) \qquad a, b > 0$$

to rearrange so that logarithms act directly on the probabilities:

$$\ln\left(\frac{1}{2}\Pr_{\text{KT}}(h_n) + \frac{1}{2}P_w^{n0}P_w^{n1}\right) = \begin{cases} \ln(1/2) + \ln \Pr_{\text{KT}}(h_n) + \ln\left(1 + \exp\left(\ln P_w^{n0} + \ln P_w^{n1} - \ln \Pr_{\text{KT}}(h_n)\right)\right) \\ \ln(1/2) + \ln P_w^{n0} + \ln P_w^{n1} + \ln\left(1 + \exp\left(\ln \Pr_{\text{KT}}(h_n) - \ln P_w^{n0} + \ln P_w^{n1}\right)\right) \end{cases}.$$

In order to avoid overflow problems, we choose the formulation for which the argument of the exponent $\exp(\ln b - \ln a)$ is as small as possible.

5.4.3.10 int CTNode::visits (void) const [inline]

The number of times this context has been visited. This is equivalent to the sum of the values in CTNode::m_count as well as to the sum of the visits of the (immediate) child nodes.

5.4.4 Friends And Related Function Documentation

5.4.4.1 friend class ContextTree [friend]

The ContextTree class is made a friend so it can access the private members of CTNode. There are several reasons for this:

- The log weighted block probability and log KT estimated block probability are calculated by the ContextTree::update() method and the CTNode::logProbWeighted() and CTNode::logProbEstimated() methods simply return these calculated values.
- This arrangement allows the ContextTree class to create/delete nodes from the context tree.

5.4.5 Field Documentation

5.4.5.1 CTNode* CTNode::m_child[2] [private]

The children of this node.

5.4.5.2 int CTNode::m_count[2] [private]

The number of zeros (CTNode::m_count[0]) and ones (CTNode::m_count[1]) in the history subsequence relevant to this node.

5.4.5.3 weight_t CTNode::m_log_kt [private]

The cached KT estimate of the block log probability for this node.

5.4.5.4 weight_t CTNode::m_log_probability [private]

The cached weighted log probability for this node.

The documentation for this class was generated from the following files:

- predict.hpp
- predict.cpp

5.5 Environment Class Reference

`#include <environment.hpp>`

Inheritance diagram for Environment:

Public Member Functions

- int actionBits () const
- percept_t getObservation (void) const
- percept_t getReward (void) const
- virtual bool isFinished (void) const
- virtual bool isValidAction (const action_t action) const
- virtual bool isValidObservation (const percept_t observation) const
- virtual bool isValidReward (const percept_t reward) const
- virtual action_t maxAction () const =0
- virtual percept_t maxObservation () const =0
- virtual percept_t maxReward () const =0
- virtual action_t minAction () const
- virtual percept_t minObservation () const
- virtual percept_t minReward () const
- int observationBits () const
- int perceptBits () const
- virtual void performAction (action_t action)=0
- virtual std::string print (void) const
- int rewardBits () const

Protected Attributes

- action_t m_action
- percept_t m_observation
- percept_t m_reward

5.5.1 Detailed Description

Base class for the various environments. Each individual environment should inherit from this class and implement the appropriate methods. In particular, the constructor should set up the environment as appropriate, including setting the initial observation and reward, as well as setting appropriate values for the configuration options:

- "agent-actions"
- "observation-bits"
- "reward-bits"

Following this, the agent and environment interact in a cyclic fashion. The agent receives the observation and reward using Environment::getObservation() and Environment::getReward() before supplying the environment with an action via Environment::performAction(). Upon receiving an action, the environment updates the observation and reward. At the beginning of each cycle, the value of Environment::isFinished() is checked. If it is true then there is no more interaction between the agent and environment and the program exits. Otherwise the interaction continues indefinitely.

5.5.2 Member Function Documentation

5.5.2.1 int Environment::actionBits ( ) const [inline]

The maximum number of bits required to represent an action.

5.5.2.2 percept_t Environment::getObservation (void) const [inline]

Returns

The current observation.

5.5.2.3 percept_t Environment::getReward (void) const [inline]

Returns

The current reward.

5.5.2.4 virtual bool Environment::isFinished (void) const [inline, virtual]

Returns

True if the environment cannot interact with the agent anymore.

5.5.2.5 bool Environment::isValidAction (const action_t *action*) const [virtual]

Checks if an action is valid.

5.5.2.6 bool Environment::isValidObservation (const percept_t *observation*) const [virtual]

Checks if an observation is valid (i.e. possible to observe).

5.5.2.7 bool Environment::isValidReward (const percept_t *reward*) const [virtual]

Checks if a reward is valid (i.e. possible to observe).

5.5.2.8 virtual action_t Environment::maxAction ( ) const [pure virtual]

The maximum possible action.

Implemented in CoinFlip, ExtendedTiger, KuhnPoker, Maze, PacMan, RelayMaze, RockPaperScissors, TicTacToe, and Tiger.

5.5.2.9 virtual percept_t Environment::maxObservation ( ) const [pure virtual]

The maximum possible observation.

Implemented in CoinFlip, ExtendedTiger, KuhnPoker, Maze, PacMan, RelayMaze, RockPaperScissors, TicTacToe, and Tiger.

5.5.2.10 virtual percept_t Environment::maxReward ( ) const [pure virtual]

The maximum possible reward.

Implemented in CoinFlip, ExtendedTiger, KuhnPoker, Maze, PacMan, RelayMaze, RockPaperScissors, TicTacToe, and Tiger.

5.5.2.11 virtual action_t Environment::minAction ( ) const [inline, virtual]

The minimum possible action.

5.5.2.12 virtual percept_t Environment::minObservation ( ) const [inline, virtual]

The minimum possible observation.

5.5.2.13 virtual percept_t Environment::minReward ( ) const [inline, virtual]

The minimum possible reward.

5.5.2.14 int Environment::observationBits ( ) const [inline]

The maximum number of bits required to represent an observation.

5.5.2.15 int Environment::perceptBits ( ) const [inline]

The maximum number of bits required to represent a percept.

5.5.2.16 virtual void Environment::performAction (action_t *action*) [pure virtual]

Receives the agent's action and calculates the new environment percept.

Implemented in CoinFlip, ExtendedTiger, KuhnPoker, Maze, PacMan, RelayMaze, RockPaperScissors, TicTacToe, and Tiger.

5.5.2.17 std::string Environment::print (void) const [virtual]

Reimplemented in CoinFlip, ExtendedTiger, KuhnPoker, Maze, PacMan, RelayMaze, RockPaperScissors, TicTacToe, and Tiger.

5.5.2.18 int Environment::rewardBits ( ) const [inline]

The maximum number of bits required to represent a reward.

5.5.3 Field Documentation

5.5.3.1 action_t Environment::m_action [protected]

The last action performed by the agent.

5.5.3.2 percept_t Environment::m_observation [protected]

The current observation.

5.5.3.3 percept_t Environment::m_reward [protected]

The current reward.

The documentation for this class was generated from the following files:

- environment.hpp
- environment.cpp

5.6 ExtendedTiger Class Reference

`#include <extendedtiger.hpp>`

Inheritance diagram for ExtendedTiger:

Collaboration diagram for ExtendedTiger:

Public Member Functions

- ExtendedTiger (options_t &options)
- virtual action_t maxAction () const
- virtual percept_t maxObservation () const
- virtual percept_t maxReward () const
- virtual void performAction (const action_t action)
- virtual std::string print () const

Private Member Functions

- void reset ()

Private Attributes

- percept_t m_gold
- double m_listen_accuracy
- bool m_sitting
- percept_t m_tiger

Static Private Attributes

- static const action_t aLeft = 1
- static const action_t aListen = 0
- static const action_t aRight = 2
- static const action_t aStand = 3
- static const double cDefaultListenAccuracy = 0.85
- static const percept_t oLeft = 1
- static const percept_t oNull = 0
- static const percept_t oRight = 2
- static const percept_t rGold = 130
- static const percept_t rInvalid = 0
- static const percept_t rListen = 100
- static const percept_t rStand = 99
- static const percept_t rTiger = 0

5.6.1 Detailed Description

The environment is a more elaborate version of Tiger. There are two doors and a stool. A tiger is hidden behind one door and a pot of gold is hidden behind the other. The agent begins each round sitting on the stool where it may either listen for the tiger (ExtendedTiger::aListen) or stand up (ExtendedTiger::aStand). Listening for the tiger results in an observation which correctly describes the tiger's whereabouts with probability ExtendedTiger::m_listen_accuracy and a reward of ExtendedTiger::rListen. Standing up result in an uninformative observation (ExtendedTiger::oNull) and a reward of ExtendedTiger::rStand. Once the agent is standing, it may open either the left or right door (ExtendedTiger::oLeft and ExtendedTiger::oRight). Doing so results in an uninformative observation (ExtendedTiger::oNull) and a reward based on what is behind the door (ExtendedTiger::rGold or ExtendedTiger::rTiger). After opening a door the agent is reseated and the tiger and gold randomly re-allocated to a door (ExtendedTiger::reset()). Attempting to open

a door while seated, to listen while standing, or to stand while already standing will result in an uninformative observation (ExtendedTiger::oNull) and a reward of ExtendedTiger::rInvalid.

Domain characteristics:

- environment: "extended-tiger"
- maximum action: 3 (2 bits)
- maximum observation: 2 (2 bits)
- maximum reward: 130 (8 bits)

Configuration options:

- tiger-listen-accuracy (optional): probability that listening (while seated) will correctly locate the door which hides the tiger. Must be a floating point value between 0.0 and 1.0 inclusive. Stored in ExtendedTiger::m_listen_accuracy. Default value is ExtendedTiger::cDefaultListenAccuracy.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 ExtendedTiger::ExtendedTiger (options_t & *options*)

5.6.3 Member Function Documentation

5.6.3.1 virtual action_t ExtendedTiger::maxAction ( ) const [inline, virtual]

The maximum possible action.

Implements Environment.

5.6.3.2 virtual percept_t ExtendedTiger::maxObservation ( ) const [inline, virtual]

The maximum possible observation.

Implements Environment.

5.6.3.3 virtual percept_t ExtendedTiger::maxReward ( ) const [inline, virtual]

The maximum possible reward.

Implements Environment.

5.6.3.4 void ExtendedTiger::performAction (const action_t *action*) [virtual]

Receives the agent's action and calculates the new environment percept.

Implements Environment.

5.6.3.5 std::string ExtendedTiger::print (void) const [virtual]

Reimplemented from Environment.

5.6.3.6 void ExtendedTiger::reset (void) [private]

Randomly place the tiger behind one door, the gold behind the other, and re-seat the agent.

5.6.4 Field Documentation

5.6.4.1 const action_t ExtendedTiger::aLeft = 1 [static, private]

Action: open left door.

5.6.4.2 const action_t ExtendedTiger::aListen = 0 [static, private]

Action: listen for tiger.

5.6.4.3 const action_t ExtendedTiger::aRight = 2 [static, private]

Action: open right door.

5.6.4.4 const action_t ExtendedTiger::aStand = 3 [static, private]

ACtion: stand up.

5.6.4.5 const double ExtendedTiger::cDefaultListenAccuracy = 0.85 [static, private]

Default value of ExtendedTiger::m_listen_accuracy.

5.6.4.6 percept_t ExtendedTiger::m_gold [private]

Observation corresponding to the door containing the gold.

5.6.4.7 double ExtendedTiger::m_listen_accuracy [private]

Probability that listening (while sitting down) correctly identifies the door which hides the tiger. Default is ExtendedTiger::cDefaultListenAccuracy.

5.6.4.8 bool ExtendedTiger::m_sitting [private]

True if the agent is sitting, false if the agent is standing.

5.6.4.9 percept_t ExtendedTiger::m_tiger [private]

Observation corresponding to the door containing the tiger.

5.6.4.10 const percept_t ExtendedTiger::oLeft = 1 [static, private]

Observation: hear tiger at left door.

5.6.4.11 const percept_t ExtendedTiger::oNull = 0 [static, private]

Uninformative observation. Received when opening a door, standing up, or listening while standing.

5.6.4.12 const percept_t ExtendedTiger::oRight = 2 [static, private]

Observation: hear tiger at right door.

5.6.4.13 const percept_t ExtendedTiger::rGold = 130 [static, private]

Reward: find gold.

5.6.4.14 const percept_t ExtendedTiger::rInvalid = 0 [static, private]

Reward: invalid action.

5.6.4.15 const percept_t ExtendedTiger::rListen = 100 [static, private]

Reward: attempt to listen while sitting.

5.6.4.16 const percept_t ExtendedTiger::rStand = 99 [static, private]

Reward: successfully stand up.

5.6.4.17 const percept_t ExtendedTiger::rTiger = 0 [static, private]

Reward: eaten by tiger.

The documentation for this class was generated from the following files:

- extendedtiger.hpp
- extendedtiger.cpp

5.7 KuhnPoker Class Reference

`#include <kuhnpoker.hpp>`

Inheritance diagram for KuhnPoker:

Collaboration diagram for KuhnPoker:

Public Member Functions

- KuhnPoker (options_t &options)
- virtual action_t maxAction () const
- virtual percept_t maxObservation () const
- virtual percept_t maxReward () const
- virtual void performAction (action_t action)
- virtual std::string print () const

Private Member Functions

- std::string cardToString (const percept_t card) const
- percept_t randomCard () const
- void reset ()

Private Attributes

- percept_t m_agent_card
- percept_t m_agent_previous_card
- action_t m_env_action
- percept_t m_env_card
- action_t m_env_previous_action
- percept_t m_env_previous_card

Static Private Attributes

- static const action_t aBet = 0
- static const action_t aPass = 1
- static const double cBetProbJack = KuhnPoker::cBetProbKing / 3.0
- static const double cBetProbKing = 0.7
- static const double cBetProbQueen = (1.0 + KuhnPoker::cBetProbKing) / 3.0
- static const percept_t oBet = 0
- static const percept_t oJack = 0
- static const percept_t oKing = 2
- static const percept_t oPass = 4
- static const percept_t oQueen = 1
- static const percept_t rBetLoss = 0
- static const percept_t rBetWin = 4
- static const percept_t rPassLoss = 1
- static const percept_t rPassWin = 3

5.7.1 Detailed Description

Kuhn Poker is a simplified, zero-sum, two player poker variant that uses a deck of three cards: a King, Queen and Jack. Whilst considerably less sophisticated than popular poker variants such as Texas Hold'em, well-known strategic concepts such as bluffing and slow-playing remain characteristic of strong play.

In this setup, the agent acts second in a series of rounds. Two actions, pass (KuhnPoker::aPass) or bet (KuhnPoker::aBet), are available to each player. A bet action requires the player to put an extra chip into play. At the beginning of each round, each player puts a chip into play. The environment (opponent) then decides whether to pass or bet; betting will win the round if the agent subsequently passes, otherwise a showdown will occur. In a showdown, the player with the highest card wins the round (i.e. King beats Queen, Queen beats Jack). If the environment (opponent) passes, the agent can either bet or pass; passing leads immediately to a showdown, whilst betting requires the environment (opponent) to either bet to force a showdown, or to pass and let the agent win the round uncontested. The winner of the round gains a reward equal to the total chips in play (KuhnPoker::rPassWin or KuhnPoker::rBetWin), the loser receives a penalty equal to the number of chips they put into play this round (KuhnPoker::rPassLoss or KuhnPoker::rBetLoss). At the end of the round, all chips are removed from play and another round begins.

Domain characteristics:

- environment: "kuhnpoker"
- maximum action: 1 (1 bit)
- maximum observation: 6 (3 bits)
- maximum reward: 4 (3 bits)

5.7.2 Constructor & Destructor Documentation

5.7.2.1 KuhnPoker::KuhnPoker (options_t & *options*)

5.7.3 Member Function Documentation

5.7.3.1 std::string KuhnPoker::cardToString (const percept_t *card*) const [private]

Turn a card observation into a human-readable string.

5.7.3.2 virtual action_t KuhnPoker::maxAction ( ) const [inline, virtual]

The maximum possible action.

Implements Environment.

5.7.3.3 virtual percept_t KuhnPoker::maxObservation ( ) const [inline, virtual]

The maximum possible observation.

Implements Environment.

5.7.3.4 virtual percept_t KuhnPoker::maxReward ( ) const [inline, virtual]

The maximum possible reward.

Implements Environment.

5.7.3.5 void KuhnPoker::performAction (action_t *action*) [virtual]

Receives the agent's action and calculates the new environment percept.

Implements Environment.

5.7.3.6 std::string KuhnPoker::print (void) const [virtual]

Reimplemented from Environment.

5.7.3.7 percept_t KuhnPoker::randomCard ( ) const [private]

Choose a card uniformly at random.

5.7.3.8 void KuhnPoker::reset (void) [private]

Begin a new round. Save necessary information for KuhnPoker::print(), deal new cards, choose environment's (opponent's) first action and compute initial observation.

5.7.4 Field Documentation

5.7.4.1 const action_t KuhnPoker::aBet = 0 [static, private]

Action: agent bets an additional token.

5.7.4.2 const action_t KuhnPoker::aPass = 1 [static, private]

Action: agent passes (does not bet).

5.7.4.3 const double KuhnPoker::cBetProbJack = KuhnPoker::cBetProbKing / 3.0 [static, private]

The probability that the environment (opponent) will bet on a jack during its initial bet.

5.7.4.4 const double KuhnPoker::cBetProbKing = 0.7 [static, private]

The probability that the environment (opponent) will bet on a king during its initial bet.

5.7.4.5 const double KuhnPoker::cBetProbQueen = (1.0 + KuhnPoker::cBetProbKing) / 3.0 [static, private]

The probability that the environment (opponent) will bet on a queen during its second bet.

5.7.4.6 percept_t KuhnPoker::m_agent_card [private]

The agent's card.

5.7.4.7 percept_t KuhnPoker::m_agent_previous_card [private]

The agent's card in the previous round. Used by KuhnPoker::print().

5.7.4.8 action_t KuhnPoker::m_env_action [private]

The enviroment's (opponent's) current action.

5.7.4.9 percept_t KuhnPoker::m_env_card [private]

The environment's (opponent's) card.

5.7.4.10 action_t KuhnPoker::m_env_previous_action [private]

The environment's (opponent's) action in the previous round. Used by KuhnPoker::print().

5.7.4.11 percept_t KuhnPoker::m_env_previous_card [private]

The environments's (opponent's) card in the previous round. Used by KuhnPoker::print().

5.7.4.12 const percept_t KuhnPoker::oBet = 0 [static, private]

Observation: the environment (opponent) bety.

5.7.4.13 const percept_t KuhnPoker::oJack = 0 [static, private]

Observation: the agent's card is a jack.

5.7.4.14 const percept_t KuhnPoker::oKing = 2 [static, private]

Observation: the agent's card is a king.

5.7.4.15 const percept_t KuhnPoker::oPass = 4 [static, private]

Observation: the environment (opponent) passed.

5.7.4.16 const percept_t KuhnPoker::oQueen = 1 [static, private]

Observation: the agent's card is a queen.

5.7.4.17 const percept_t KuhnPoker::rBetLoss = 0 [static, private]

Reward: agent lost and bet.

5.7.4.18 const percept_t KuhnPoker::rBetWin = 4 [static, private]

Reward: agent won and environment (opponent) bet.

5.7.4.19 const percept_t KuhnPoker::rPassLoss = 1 [static, private]

Reward: agent lost and passed.

5.7.4.20 const percept_t KuhnPoker::rPassWin = 3 [static, private]

Reward: agent won and environment (opponent) passed.

The documentation for this class was generated from the following files:

- kuhnpoker.hpp

- kuhnpoker.cpp

5.8 Maze Class Reference

`#include <maze.hpp>`

Inheritance diagram for Maze:

Collaboration diagram for Maze:

Public Member Functions

- virtual action_t maxAction () const
- virtual percept_t maxObservation () const
- virtual percept_t maxReward () const
- Maze (options_t &options)
- virtual void performAction (const action_t action)
- virtual std::string print () const
- ~Maze ()

Private Types

- enum { cUninformative, cWalls, cCoordinates }

Private Member Functions

- void calculateObservation ()
- void configure (options_t &options)
- void teleportAgent ()

Private Attributes

- int m_col
- int m_max_reward
- char ** m_maze_layout
- int ** m_maze_rewards
- int m_num_cols
- int m_num_rows
- enum Maze:: { ... } m_obs_encoding
- int m_row
- bool m_teleported
- bool m_wall_collision

Static Private Attributes

- static const action_t aDown = 3
- static const action_t aLeft = 0
- static const action_t aRight = 2
- static const action_t aUp = 1
- static const char cEmpty = '&'
- static const char cTeleportFrom = '!'
- static const char cTeleportTo = '*'
- static const char cWall = '@'
- static const percept_t oDownWall = 8
- static const percept_t oLeftWall = 1
- static const percept_t oNull = 0
- static const percept_t oRightWall = 4
- static const percept_t oUpWall = 2

5.8.1 Detailed Description

A two-dimensional maze environment. The agent is able to move through the maze in each of the four cardinal directions (Maze::aLeft, Maze::aUp, Maze::aRight, Maze::aDown).The user is able to specify (via a connfiguration file) the dimensions, layout, and rewards of the maze as well as the type of observations given to the agent. In particular, the maze is a certain number of rows high and columns wide, each cell in the maze is of a certain type and has a certain reward.

The type of each cell determines what happens to the agent when it attempts to move into the cell:

- Wall (Maze::cWall): represents an impassable cell, attempting to move into a wall will result in the agent remaining at it's current position.
- Empty (Maze::cEmpty): an empty cell through which the agent can freely pass.
- Teleport from (Maze::cTeleportFrom): represents a cell which, upon entry, will randomly teleport the agent to another cell of type Maze::cTeleportTo.
- Teleport to (Maze::cTeleportTo): A cell which can be teleported to. Otherwise, the cell acts identically to an empty cell. The maze MUST contain at least one of these cells.

The reward for each cell describes the reward received by the agent when it attempts to move into that cell. For example, when the agent attempts to move into a wall square, it gets the reward from the wall cell despite the fact that it does not end its turn in that cell. Similarly, when the agent moves into a Maze::cTeleportFrom cell it gets the reward from that cell rather than from the cell it will be teleported to.

Finally, the user can choose one of several observation encodings to give to the agent:

- Uninformative (Maze::cUninformative): The observation is a single unchanging value.
- Walls (Maze::cWalls): The observation encodes the presence of walls in adjacent squares. This is encoded as a sum of the flags: Maze::oDownWall, Maze::oLeftWall, Maze::oRightWall, Maze::oUpWall, each of which indicates the presence of a wall in the corresponding direction.
- Coordinates (Maze::cCoordinates): The observation gives the coordinates of the cell occupied by the agent. This is encoded as row * num_cols + col.

Domain characteristic:

- environment: "maze"
- maximum action: 3 (2 bits)
- maximum observation:
  - Uninformative: 0 (1 bit)
  - Walls: 15 (4 bits)
  - Coordinates: Maze::m_num_rows * Maze::m_num_cols - 1
- maximum reward: maximum value in maze rewards (Maze::m_maze_rewards)

Configuration options:

- maze-num-rows: the number of rows in the maze.
- maze-num-cols: the number of columns in the maze.
- maze-rewards#: comma-separated list of rewards for each square in row #. If the agent enters (or attempts to enter) a particular square it receives the corresponding reward. 1 <= # <= maze-num-rows.
- maze-layout#: The layout of row # of the maze (1 <= # <= maze-num-rows). Contains maze-num-cols symbols as follows:
  - Maze::cWall: indicates the square cannot be entered (a.k.a. "a wall").
  - Maze::cTeleportTo: indicates the square can be entered and teleported to.
  - Maze::cEmpty: indicates the square can be entered but not teleported to.
  - Maze::cTeleportFrom: indicates the square can be entered, and that doing so will randomly teleport the agent to a Maze::cTeleportTo square before the next turn (the agent receives the reward before the teleportation occurs).
- maze-observation-encoding: Specifies the type of observations the agent recieves
  - uninformative: the agent receives the same observation each cycle.
  - walls: the agent receives an observation specifying whether there are walls "@" above, below, left, or right of its current position.
  - coordinates: the observation specifies the coordinates of the agent in the maze.

5.8.2 Member Enumeration Documentation

5.8.2.1 anonymous enum `[private]`

The type and encoding of observations provided to the agent. Default is uninformative observations.

Enumerator:

cUninformative Unchanging observation.

cWalls Observe presence of adjacent walls.

cCoordinates Observe coordinates of current square.

5.8.3 Constructor & Destructor Documentation

5.8.3.1 Maze::Maze (options_t & *options*)

5.8.3.2 Maze::~Maze ( )

Frees memory allocated to store Maze::m_maze_rewards and Maze::m_maze_layout.

5.8.4 Member Function Documentation

5.8.4.1 void Maze::calculateObservation ( ) [private]

Determine the observation to give to the agent based on its current location (Maze::m_row and Maze::m_col) and the observation type (Maze::m_obs_type). Store the observation in Maze::m_observation.

5.8.4.2 void Maze::configure (options_t & *options*) [private]

Configure the maze based on the configuration options. May exit if the configuration is not validly formatted.

5.8.4.3 virtual action_t Maze::maxAction ( ) const [inline, virtual]

The maximum possible action.

Implements Environment.

5.8.4.4 percept_t Maze::maxObservation ( ) const [virtual]

The maximum observation that can be given to the agent. Depends on the observation type (Maze::m_obs_type) and (potentially) the dimensions of the maze (Maze::m_num_rows and Maze::m_num_cols).

Implements Environment.

5.8.4.5 virtual percept_t Maze::maxReward ( ) const [inline, virtual]

The maximum reward that can be received by the agent. This is simply the maximum value in array Maze::m_maze_rewards (cached in Maze::m_max_reward).

Implements Environment.

5.8.4.6 void Maze::performAction (const action_t *action*) [virtual]

Receives the agent's action and calculates the new environment percept.

Implements Environment.

5.8.4.7 std::string Maze::print (void) const [virtual]

Print the current state of the environment, including the current location, observation, reward, and maze layout.

Reimplemented from Environment.

5.8.4.8 void Maze::teleportAgent ( ) [private]

Randomly place the agent at any Maze::cTeleportTo location.

5.8.5 Field Documentation

5.8.5.1 const action_t Maze::aDown = 3 [static, private]

Action: the agent moves down.

5.8.5.2 const action_t Maze::aLeft = 0 [static, private]

Action: the agent moves left.

5.8.5.3 const action_t Maze::aRight = 2 [static, private]

Action: the agent moves right.

5.8.5.4 const action_t Maze::aUp = 1 [static, private]

Action: the agent moves up.

5.8.5.5 const char Maze::cEmpty = '&' [static, private]

Represents an empty square in the maze which cannot be teleported to or from.

5.8.5.6 const char Maze::cTeleportFrom = '!' [static, private]

Represents an empty square in the maze which, upon entry, causes the agent to teleport to another square of type Maze::cTeleportTo.

5.8.5.7 const char Maze::cTeleportTo = '∗' [static, private]

Represents an empty square in the maze which can be teleported to.

5.8.5.8 const char Maze::cWall = '@' [static, private]

Represents an impassable square in the maze.

5.8.5.9 int Maze::m_col [private]

The column occupied by the agent.

5.8.5.10 int Maze::m_max_reward [private]

The maximum possible reward in a single cycle. This is equivalent to the maximum value in Maze::m_maze_rewards.

5.8.5.11 char** Maze::m_maze_layout [private]

Stores the layout of the maze. In particular, Maze::m_maze_layout[r][c] specifies the type of square in row r and column c. Different types of squares include Maze::cWall, Maze::cTeleportTo, Maze::cTeleportFrom and Maze::cEmpty. Row # of the layout array is set via the (mandatory) configuration option "maze-layout#" (where # is replaced with the row number).

5.8.5.12 int** Maze::m_maze_rewards [private]

Stores the reward for each square in the maze. In particular, Maze::m_maze_rewards[r][c] is the reward given to the agent when it attempts to move into the square at the intersection of row r and column c (regardless of whether it is actually able to move into the desired square). All rewards are nonnegative. Row # of the reward array is set via the (mandatory) configuration option "maze-rewards#" (where # is replaced with the row number).

5.8.5.13 int Maze::m_num_cols [private]

The number of columns in the maze. Set via required configuration option "maze-num-cols".

5.8.5.14 int Maze::m_num_rows [private]

The number of rows in the maze. Set via required configuration option "maze-num-rows".

5.8.5.15 enum { ... } Maze::m_obs_encoding [private]

The type and encoding of observations provided to the agent. Default is uninformative observations.

5.8.5.16 int Maze::m_row [private]

The row occupied by the agent.

5.8.5.17 bool Maze::m_teleported [private]

Indicates whether the last action caused the agent to teleport.

5.8.5.18 bool Maze::m_wall_collision [private]

Indicates whether the last action caused the agent to collide with a wall (Maze::cWall).

5.8.5.19 const percept_t Maze::oDownWall = 8 [static, private]

Observation: A wall is present immediately below the agent.

5.8.5.20 const percept_t Maze::oLeftWall = 1 [static, private]

Observation: A wall is present immediately to the left of the agent.

5.8.5.21 const percept_t Maze::oNull = 0 [static, private]

Uninformative observation used when Maze::m_obs_encoding is set to give uninformative observations.

5.8.5.22 const percept_t Maze::oRightWall = 4 [static, private]

Observation: A wall is present immediately to the right of the agent.

5.8.5.23 const percept_t Maze::oUpWall = 2 [static, private]

Observation: A wall is present immediately above the agent.

The documentation for this class was generated from the following files:

- maze.hpp
- maze.cpp

5.9 ModelUndo Class Reference

`#include <agent.hpp>`

Public Member Functions

- age_t age (void) const
- size_t historySize (void) const
- update_t lastUpdate (void) const
- ModelUndo (const Agent &agent)
- reward_t reward (void) const

Private Attributes

- age_t m_age
- size_t m_history_size
- update_t m_last_update
- reward_t m_reward

5.9.1 Detailed Description

The ModelUndo class is used to store the information required to restore an agent to a copy of itself from a previous time cycle. In particular, it is sufficient to back up the following attributes:

- Agent::age()
- Agent::reward()

- Agent::historySize()
- Agent::lastUpdatePercept()

5.9.2 Constructor & Destructor Documentation

5.9.2.1 ModelUndo::ModelUndo (const Agent & *agent*)

Extracts the information required to restore an agent to its current state.

Parameters

agent The agent whose state we wish to "save".

5.9.3 Member Function Documentation

5.9.3.1 age_t ModelUndo::age (void) const [inline]

The age of the agent (Agent::age()) at the time it was saved.

5.9.3.2 size_t ModelUndo::historySize (void) const [inline]

The size of the agents history size (Agent::historySize()) at the time it was saved.

5.9.3.3 update_t ModelUndo::lastUpdate (void) const [inline]

The status of the agents last update (Agent::lastUpdatePercept()) at the time it was saved.

5.9.3.4 reward_t ModelUndo::reward (void) const [inline]

The total reward of the agent (Agent::reward()) at the time it was saved.

5.9.4 Field Documentation

5.9.4.1 age_t ModelUndo::m_age [private]

5.9.4.2 size_t ModelUndo::m_history_size [private]

5.9.4.3 update_t ModelUndo::m_last_update [private]

5.9.4.4 reward_t ModelUndo::m_reward [private]

The documentation for this class was generated from the following files:

- agent.hpp
- agent.cpp

5.10 PacMan Class Reference

`#include <pacman.hpp>`

Inheritance diagram for PacMan:

Collaboration diagram for PacMan:

Public Member Functions

- virtual action_t maxAction () const
- virtual percept_t maxObservation () const
- virtual percept_t maxReward () const
- PacMan (options_t &options)
- virtual void performAction (action_t action)
- virtual std::string print () const

Private Member Functions

- int binaryToDecimal (bool binary[ ])
- bool findPacman (int *ghostX, int *ghostY, int *sniff)
- void ghostPursuitMove (int *ghostX, int *ghostY, int *covers, char ghost)
- void ghostRandomMove (int *ghostX, int *ghostY, int *covers, char ghost)
- bool isValidGhostMove (int x, int y)
- int manhattanDistance (int x1, int y1, int x2, int y2)
- void moveGhostAndUpdateReward (int *ghostX, int *ghostY, int *sniff, int *covers, char ghost)
- void movePacmanAndUpdateReward (int x, int y)
- void resetEpisode ()
- void resetGhost (char ghost)
- void updateObservation ()

Private Attributes

- int aGhostCovering
- int aGhostX
- int aGhostY
- int bGhostCovering
- int bGhostX
- int bGhostY
- bool binaryObservation [16]
- int cGhostCovering
- int cGhostX
- int cGhostY
- int dGhostCovering
- int dGhostX
- int dGhostY
- std::string map [19]
- int pacmanX
- int pacmanY
- int pelletCount

- bool poweredUp
- int powerLeft
- bool reset
- int resets
- int sniffA
- int sniffB
- int sniffC
- int sniffD
- int timestep

5.10.1 Detailed Description

This domain is a partially observable version of the classic PacMan game. The agent must navigate a 17x17 maze and eat the food pellets that are distributed across the maze. Four ghosts roam the maze. They move initially at random, until there is a Manhattan distance of 5 between them and PacMan, whereupon they will aggressively pursue PacMan for a short duration. The maze structure and game are the same as the original arcade game, however the PacMan agent is hampered by partial observability. PacMan is unaware of the maze structure and only receives a 4-bit observation describing the wall configuration at its current location. It also does not know the exact location of the ghosts, receiving only 4-bit observations indicating whethe a ghost is visible (via direct line of sight) in each of the four cardinal directions. In addition, the location of the food pellets is unknown except for a 3-bit observation that indicates whether food can be smelt within a Manhattan distance of 2, 3 or 4 from PacMan's location, and another 4-bit observation indicating whether there is food in its direct line of sight. A final single bit indicates whether PacMan is under the effects of a power pill. At the start of each episode, a food pellet is placed down with probability 0.5 at every empty location on the grid. The agent receives a penalty of 1 for each movement action, a penalty of 10 for running into a wall, a reward of 10 for each food pellet eaten, a penalty of 50 if it is caught by a ghost, and a reward of 100 for collecting all the food. If multiple such events occur, then the total reward is cumulative, i.e. running into a wall and being caught would give a penalty of 60. The episode resets if the agent is caught or if it collects all the food.

Domain characteristics:

- maximum action: 3
- observation bits: 16
- reward bits: 8

5.10.2 Constructor & Destructor Documentation

5.10.2.1 PacMan::PacMan (options_t & *options*)

5.10.3 Member Function Documentation

5.10.3.1 int PacMan::binaryToDecimal (bool *binary[ ]*) [private]

5.10.3.2 bool PacMan::findPacman (int * *ghostX*, int * *ghostY*, int * *sniff*) [private]

5.10.3.3 void PacMan::ghostPursuitMove (int * *ghostX*, int * *ghostY*, int * *covers*, char *ghost*) [private]

5.10.3.4 void PacMan::ghostRandomMove (int * *ghostX*, int * *ghostY*, int * *covers*, char *ghost*) [private]

5.10.3.5 bool PacMan::isValidGhostMove (int *x*, int *y*) [private]

5.10.3.6 int PacMan::manhattanDistance (int *x1*, int *y1*, int *x2*, int *y2*) [private]

5.10.3.7 virtual action_t PacMan::maxAction ( ) const [inline, virtual]

The maximum possible action.

Implements Environment.

5.10.3.8 virtual percept_t PacMan::maxObservation ( ) const [inline, virtual]

The maximum possible observation.

Implements Environment.

5.10.3.9 virtual percept_t PacMan::maxReward ( ) const [inline, virtual]

The maximum possible reward.

Implements Environment.

5.10.3.10 void PacMan::moveGhostAndUpdateReward (int * *ghostX*, int * *ghostY*, int * *sniff*, int * *covers*, char *ghost*) [private]

5.10.3.11 void PacMan::movePacmanAndUpdateReward (int *x*, int *y*) [private]

5.10.3.12 void PacMan::performAction (action_t *action*) [virtual]

Receives the agent's action and calculates the new environment percept.

Implements Environment.

5.10.3.13 std::string PacMan::print (void) const [virtual]

Reimplemented from Environment.

5.10.3.14 void PacMan::resetEpisode ( ) [private]

5.10.3.15 void PacMan::resetGhost (char *ghost*) [private]

5.10.3.16 void PacMan::updateObservation ( ) [private]

5.10.4 Field Documentation

5.10.4.1 int PacMan::aGhostCovering [private]

5.10.4.2 int PacMan::aGhostX [private]

5.10.4.3 int PacMan::aGhostY [private]

5.10.4.4 int PacMan::bGhostCovering [private]

5.10.4.5 int PacMan::bGhostX [private]

5.10.4.6 int PacMan::bGhostY [private]

5.10.4.7 bool PacMan::binaryObservation[16] [private]

5.10.4.8 int PacMan::cGhostCovering [private]

5.10.4.9 int PacMan::cGhostX [private]

5.10.4.10 int PacMan::cGhostY [private]

5.10.4.11 int PacMan::dGhostCovering [private]

5.10.4.12 int PacMan::dGhostX [private]

5.10.4.13 int PacMan::dGhostY [private]

5.10.4.14 std::string PacMan::map[19] [private]

5.10.4.15 int PacMan::pacmanX [private]

5.10.4.16 int PacMan::pacmanY [private]

5.10.4.17 int PacMan::pelletCount [private]

5.10.4.18 bool PacMan::poweredUp [private]

5.10.4.19 int PacMan::powerLeft [private]

5.10.4.20 bool PacMan::reset [private]

5.10.4.21 int PacMan::resets [private]

5.10.4.22 int PacMan::sniffA [private]

5.10.4.23 int PacMan::sniffB [private]

5.10.4.24 int PacMan::sniffC [private]

5.10.4.25 int PacMan::sniffD [private]

5.10.4.26 int PacMan::timestep [private]

The documentation for this class was generated from the following files:

- pacman.hpp
- pacman.cpp

5.11 RelayMaze Class Reference

`#include <relay-maze.hpp>`

Inheritance diagram for RelayMaze:

Collaboration diagram for RelayMaze:

Public Member Functions

- virtual action_t maxAction () const
- virtual percept_t maxObservation () const
- virtual percept_t maxReward () const
- virtual void performAction (const action_t action)
- virtual std::string print () const
- RelayMaze (options_t &options)

Protected Member Functions

- void teleportAgent ()

Protected Attributes

- int m_col
- bool m_relay_flag
- int m_row
- int m_size
- bool m_wall_collision

Static Protected Attributes

- static const action_t aDown = 3
- static const action_t aLeft = 0
- static const action_t aRight = 2
- static const action_t aUp = 1
- static const reward_t rGoalReward = 0
- static const reward_t rMoveReward = 99
- static const reward_t rRelayReward = 110
- static const reward_t rWallReward = 90

5.11.1 Constructor & Destructor Documentation

5.11.1.1 RelayMaze::RelayMaze (options_t & *options*)

5.11.2 Member Function Documentation

5.11.2.1 virtual action_t RelayMaze::maxAction ( ) const [inline, virtual]

The maximum possible action.

Implements Environment.

5.11.2.2 percept_t RelayMaze::maxObservation ( ) const [virtual]

The maximum observation that can be given to the agent.

Implements Environment.

5.11.2.3 virtual percept_t RelayMaze::maxReward ( ) const [inline, virtual]

The maximum reward that can be received by the agent.

Implements Environment.

5.11.2.4 void RelayMaze::performAction (const action_t *action*) [virtual]

Receives the agent's action and calculates the new environment percept.

Implements Environment.

5.11.2.5 std::string RelayMaze::print (void) const [virtual]

Print the current state of the environment, including the current location, observation, reward, and maze layout.

Reimplemented from Environment.

5.11.2.6 void RelayMaze::teleportAgent ( ) [protected]

Randomly place the agent at any Maze::cTeleportTo location.

5.11.3 Field Documentation

5.11.3.1 const action_t RelayMaze::aDown = 3 [static, protected]

Action: the agent moves down.

5.11.3.2 const action_t RelayMaze::aLeft = 0 [static, protected]

Action: the agent moves left.

5.11.3.3 const action_t RelayMaze::aRight = 2 [static, protected]

Action: the agent moves right.

5.11.3.4 const action_t RelayMaze::aUp = 1 [static, protected]

Action: the agent moves up.

5.11.3.5 int RelayMaze::m_col [protected]

The column occupied by the agent.

5.11.3.6 bool RelayMaze::m_relay_flag [protected]

Indicates whether the agent has passed through the relay cell.

5.11.3.7 int RelayMaze::m_row [protected]

The row occupied by the agent.

5.11.3.8 int RelayMaze::m_size [protected]

The size (number of rows and columns) of the maze.

5.11.3.9 bool RelayMaze::m_wall_collision [protected]

Indicates whether the last action caused the agent to collide with a wall (Maze::cWall).

5.11.3.10 const reward_t RelayMaze::rGoalReward = 0 [static, protected]

5.11.3.11 const reward_t RelayMaze::rMoveReward = 99 [static, protected]

5.11.3.12 const reward_t RelayMaze::rRelayReward = 110 [static, protected]

5.11.3.13 const reward_t RelayMaze::rWallReward = 90 [static, protected]

The documentation for this class was generated from the following files:

- relay-maze.hpp
- relay-maze.cpp

5.12 RockPaperScissors Class Reference

`#include <rock-paper-scissors.hpp>`

Inheritance diagram for RockPaperScissors:

Collaboration diagram for RockPaperScissors:

Public Member Functions

- virtual action_t maxAction () const
- virtual percept_t maxObservation () const
- virtual percept_t maxReward () const
- virtual void performAction (const action_t action)
- virtual std::string print () const
- RockPaperScissors (options_t &options)

Static Private Attributes

- static const action_t aPaper = 1
- static const action_t aRock = 0
- static const action_t aScissors = 2
- static const percept_t oPaper = 1
- static const percept_t oRock = 0
- static const percept_t oScissors = 2
- static const percept_t rDraw = 1
- static const percept_t rLose = 0
- static const percept_t rWin = 2

5.12.1 Detailed Description

The agent repeatedly plays Rock-Paper-Scissor against an opponent that has a slight, predictable bias in its strategy. If the opponent has won a round by playing rock on the previous cycle, it will always play rock at the next time step; otherwise it will pick an action uniformly at random. The agent's observation is the most recently chosen action of the opponent. It receives a reward of RockPaperScissors::rWin for a win, RockPaperScissors::rDraw for a draw and RockPaperScissors::rLose for a loss.

Domain characteristics:

- environment: "rock-paper-scissors"
- maximum action: 2 (2 bits)
- maximum observation: 2 (2 bits)
- maximum reward: 2 (2 bits)

5.12.2 Constructor & Destructor Documentation

5.12.2.1 RockPaperScissors::RockPaperScissors (options_t & *options*)

5.12.3 Member Function Documentation

5.12.3.1 virtual action_t RockPaperScissors::maxAction ( ) const [inline, virtual]

The maximum possible action.

Implements Environment.

5.12.3.2 virtual percept_t RockPaperScissors::maxObservation ( ) const [inline, virtual]

The maximum possible observation.

Implements Environment.

5.12.3.3 virtual percept_t RockPaperScissors::maxReward ( ) const [inline, virtual]

The maximum possible reward.

Implements Environment.

5.12.3.4 void RockPaperScissors::performAction (const action_t *action*) [virtual]

Receives the agent's action and calculates the new environment percept.

Implements Environment.

5.12.3.5 std::string RockPaperScissors::print (void) const [virtual]

Reimplemented from Environment.

5.12.4 Field Documentation

5.12.4.1 const action_t RockPaperScissors::aPaper = 1 [static, private]

5.12.4.2 const action_t RockPaperScissors::aRock = 0 [static, private]

5.12.4.3 const action_t RockPaperScissors::aScissors = 2 [static, private]

5.12.4.4 const percept_t RockPaperScissors::oPaper = 1 [static, private]

5.12.4.5 const percept_t RockPaperScissors::oRock = 0 [static, private]

5.12.4.6 const percept_t RockPaperScissors::oScissors = 2 [static, private]

5.12.4.7 const percept_t RockPaperScissors::rDraw = 1 [static, private]

5.12.4.8 const percept_t RockPaperScissors::rLose = 0 [static, private]

5.12.4.9 const percept_t RockPaperScissors::rWin = 2 [static, private]

The documentation for this class was generated from the following files:

- rock-paper-scissors.hpp
- rock-paper-scissors.cpp

5.13 SearchNode Class Reference

`#include <search.hpp>`

Collaboration diagram for SearchNode:

Public Member Functions

- SearchNode * child (const interaction_t child_index) const
- reward_t expectation (void) const
- reward_t sample (Agent &agent, const int horizon)
- SearchNode (const nodetype_t nodetype)
- action_t selectAction (Agent const &agent)
- visits_t visits (void) const
- ~SearchNode (void)

Private Attributes

- child_map_t m_child
- double m_mean
- nodetype_t m_type
- visits_t m_visits

5.13.1 Detailed Description

Represents a node in the Monte Carlo search tree. The nodes in the search tree represent simulated actions and percepts between an agent following a UCB policy and a generative model of the environment represented by a context tree. The purpose of the tree is to determine the expected reward of the available actions through sampling. Sampling proceeds several time steps into the future according to the size of the agent's horizon (Agent::horizon())

The nodes are one of two types (nodetype_t), decision nodes are those whose children represent actions from the agent and chance nodes are those whose children represent percepts from the environment. Each SearchNode maintains several bits of information

- The current value of the sampled expected reward (SearchNode::m_mean, SearchNode::expectation()).
- The number of times the node has been visited during the sampling (SearchNode::m_visits, SearchNode::visits()).
- The type of the node (SearchNode::m_type).
- The children of the node (SearchNode::m_child, SearchNode::child()). The children are stored in a map of type child_map_t and are indexed by actions (decision node) or percepts (chance node).

The SearchNode::sample() function is used to sample from the current node and the SearchNode::selectAction() is used to select an action according to the UCB policy.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 SearchNode::SearchNode (const nodetype_t *nodetype*)

Create and initialise a new search node of a specific type.

5.13.2.2 SearchNode::~SearchNode (void)

Destroy all child nodes.

5.13.3 Member Function Documentation

5.13.3.1 SearchNode * SearchNode::child (const interaction_t *child_index*) const

Attempts to access the child node with a certain index.

Parameters

child_index The index of the child node. This corresponds to an action if the node is a decision node or a percept if the node is a chance node.

Returns

A pointer to the child node if it exists, otherwise return NULL.

5.13.3.2 reward_t SearchNode::expectation (void) const `[inline]`

Returns

The sampled expected reward from this node.

5.13.3.3 reward_t SearchNode::sample (Agent & *agent,* const int *horizon*)

Perform a single sample from this node.

Parameters

agent The agent which is doing the sampling.

horizon How many cycles into the future to sample.

Returns

The accumulated reward from this sample.

5.13.3.4 action_t SearchNode::selectAction (Agent const & *agent*)

Determine which action to sample according to the UCB policy.

Parameters

agent The agent which is doing the sampling.

Returns

The selected action.

5.13.3.5 visits_t SearchNode::visits (void) const [inline]

Returns

The number of times this node has been visited.

5.13.4 Field Documentation

5.13.4.1 child_map_t SearchNode::m_child [private]

The children of this node. Each corresponds to an action if this is a decision node or to a percept if this is a chance node.

5.13.4.2 double SearchNode::m_mean [private]

The sampled expected reward of this node.

5.13.4.3 nodetype_t SearchNode::m_type [private]

The type of this node indicates whether it's children represent actions (decision node) or percepts (chance node).

5.13.4.4 visits_t SearchNode::m_visits [private]

The number of times this node has been visited.

The documentation for this class was generated from the following files:

- search.hpp
- search.cpp

5.14 TicTacToe Class Reference

`#include <tictactoe.hpp>`

Inheritance diagram for TicTacToe:

Collaboration diagram for TicTacToe:

Public Member Functions

- virtual action_t maxAction () const
- virtual percept_t maxObservation () const
- virtual percept_t maxReward () const
- virtual void performAction (const action_t action)
- virtual std::string print () const
- TicTacToe (options_t &options)

Private Member Functions

- bool checkWin ()
- void computeObservation ()
- void reset ()

Private Attributes

- int m_actions_since_reset
- percept_t m_board [3][3]

Static Private Attributes

- static const percept_t oAgent = 1
- static const percept_t oEmpty = 0
- static const percept_t oEnv = 2
- static const percept_t rDraw = 4
- static const percept_t rInvalid = 0
- static const percept_t rLoss = 1
- static const percept_t rNull = 3
- static const percept_t rWin = 5

5.14.1 Detailed Description

In this domain, the agent plays repeated games of TicTacToe against an opponent who moves randomly. If the agent wins the game, it receives a reward of 2. If there is a draw, the agent receives a reward of 1. A loss penalizes the agent by -2. If the agent makes an illegal move, by moving on top of an already filled square, then it receives a reward of -3. A legal move that does not end the game earns no reward. An illegal reward causes the game to restart.

Domain characteristics:

- environment: "tictactoe"
- maximum action: 8 (4 bits)
- maximum observation: 174672 (18 bits)
  - 174672 (decimal) = 101010101010101010 (binary)
- maximum reward: 5 (3 bits)

5.14.2 Constructor & Destructor Documentation

5.14.2.1 TicTacToe::TicTacToe (options_t & *options*)

5.14.3 Member Function Documentation

5.14.3.1 bool TicTacToe::checkWin ( ) [private]

Check if either player has won the game.

5.14.3.2 void TicTacToe::computeObservation ( ) [private]

Encodes the state of each square into an overall observation and saves the result in TicTacToe::m_observation. Each cell corresponds to two bits.

5.14.3.3 virtual action_t TicTacToe::maxAction ( ) const [inline, virtual]

The maximum possible action.

Implements Environment.

5.14.3.4 virtual percept_t TicTacToe::maxObservation ( ) const [inline, virtual]

The maximum possible observation.

Implements Environment.

5.14.3.5 virtual percept_t TicTacToe::maxReward ( ) const [inline, virtual]

The maximum possible reward.

Implements Environment.

5.14.3.6 void TicTacToe::performAction (const action_t *action*) [virtual]

Receives the agent's action and calculates the new environment percept.

Implements Environment.

5.14.3.7 std::string TicTacToe::print (void) const [virtual]

Reimplemented from Environment.

5.14.3.8 void TicTacToe::reset (void) [private]

Begin a new game.

5.14.4 Field Documentation

5.14.4.1 int TicTacToe::m_actions_since_reset [private]

The number of agent actions since the last reset.

5.14.4.2 percept_t TicTacToe::m_board[3][3] [private]

Stores the tictactoe board.

5.14.4.3 const percept_t TicTacToe::oAgent = 1 [static, private]

Observation: cell occupied by agent.

5.14.4.4 const percept_t TicTacToe::oEmpty = 0 [static, private]

Observation: empty cell.

5.14.4.5 const percept_t TicTacToe::oEnv = 2 [static, private]

Observation: cell occupied by environment (opponent).

5.14.4.6 const percept_t TicTacToe::rDraw = 4 [static, private]

Reward: agent drew the game.

5.14.4.7 const percept_t TicTacToe::rInvalid = 0 [static, private]

Reward: agent performed an invalid move.

5.14.4.8 const percept_t TicTacToe::rLoss = 1 [static, private]

Reward: agent lost the game.

5.14.4.9 const percept_t TicTacToe::rNull = 3 [static, private]

Reward: agent performed a valid move, game continues.

5.14.4.10 const percept_t TicTacToe::rWin = 5 [static, private]

Reward: agent won the game.

The documentation for this class was generated from the following files:

- tictactoe.hpp
- tictactoe.cpp

5.15 Tiger Class Reference

`#include <tiger.hpp>`

Inheritance diagram for Tiger:

Collaboration diagram for Tiger:

Public Member Functions

- virtual action_t maxAction () const
- virtual percept_t maxObservation () const
- virtual percept_t maxReward () const
- virtual void performAction (const action_t action)
- virtual std::string print () const
- Tiger (options_t &options)

Private Member Functions

- void placeTiger ()

Private Attributes

- percept_t m_gold
- double m_listen_accuracy
- percept_t m_tiger

Static Private Attributes

- static const action_t aLeft = 1
- static const action_t aListen = 0
- static const action_t aRight = 2
- static const double cDefaultListenAccuracy = 0.85
- static const percept_t oLeft = 1
- static const percept_t oNull = 0
- static const percept_t oRight = 2
- static const percept_t rEaten = 0
- static const percept_t rGold = 110
- static const percept_t rListen = 99

5.15.1 Detailed Description

The environment dynamics are as follows: a tiger and a pot of gold are hidden behind one of two doors. Initially the agent starts facing both doors. The agent has a choice of one of three actions: listen, open the left door, or open the right door. If the agent opens the door hiding the tiger, it suffers a -100 penalty. If it opens the door with the pot of gold, it receives a reward of 10. If the agent performs the listen action, it receives a penalty of -1 and an observation that correctly describes where the tiger is with 0.85 probability.

Domain characteristics:

- environment: "tiger"
- maximum action: 2 (2 bits)
- maximum observation: 2 (2 bits)
- maximum reward: 110 (7 bits)

Configuration options:

- tiger-listen-accuracy (optional): the probability that listening for the tiger results in the correct observation of the tiger's whereabouts (Tiger::m_listen_accuracy). Must be a floating point number between 0.0 and 1.0 inclusive. Default value is Tiger::cDefaultListenAccuracy.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 Tiger::Tiger (options_t & *options*)

5.15.3 Member Function Documentation

5.15.3.1 virtual action_t Tiger::maxAction ( ) const [inline, virtual]

The maximum possible action.

Implements Environment.

5.15.3.2 virtual percept_t Tiger::maxObservation ( ) const [inline, virtual]

The maximum possible observation.

Implements Environment.

5.15.3.3 virtual percept_t Tiger::maxReward ( ) const [inline, virtual]

The maximum possible reward.

Implements Environment.

5.15.3.4 void Tiger::performAction (const action_t *action*) [virtual]

Receives the agent's action and calculates the new environment percept.

Implements Environment.

5.15.3.5 void Tiger::placeTiger ( ) [private]

Randomly place the tiger behind one door and the gold behind the other.

5.15.3.6 std::string Tiger::print (void) const [virtual]

Reimplemented from Environment.

5.15.4 Field Documentation

5.15.4.1 const action_t Tiger::aLeft = 1 [static, private]

Action: open left door.

5.15.4.2 const action_t Tiger::aListen = 0 [static, private]

Action: listen for the tiger.

5.15.4.3 const action_t Tiger::aRight = 2 [static, private]

Action: open right door.

5.15.4.4 const double Tiger::cDefaultListenAccuracy = 0.85 [static, private]

Default value for Tiger::m_listen_accuracy.

5.15.4.5 percept_t Tiger::m_gold [private]

The percept corresponding to the door which hides the gold.

5.15.4.6 double Tiger::m_listen_accuracy [private]

The accuracy of the listen action. Default value is Tiger::cDefaultListenAccuracy.

5.15.4.7 percept_t Tiger::m_tiger [private]

The percept corresponding to the door which hides the tiger.

5.15.4.8 const percept_t Tiger::oLeft = 1 [static, private]

Observation: hear tiger at left door.

5.15.4.9 const percept_t Tiger::oNull = 0 [static, private]

Observation: given when opening a door.

5.15.4.10 const percept_t Tiger::oRight = 2 [static, private]

Observation: hear tiger at right door.

5.15.4.11 const percept_t Tiger::rEaten = 0 [static, private]

Reward: eaten by tiger.

5.15.4.12 const percept_t Tiger::rGold = 110 [static, private]

Reward: found gold.

5.15.4.13 const percept_t Tiger::rListen = 99 [static, private]

Reward: for listening.

The documentation for this class was generated from the following files:

- tiger.hpp
- tiger.cpp

Chapter 6

File Documentation

6.1 agent.cpp File Reference

```
#include <cassert>
#include <iostream>
#include "agent.hpp"
#include "predict.hpp"
#include "search.hpp"
#include "util.hpp"
```

Include dependency graph for agent.cpp:

6.2 agent.hpp File Reference

```
#include <iostream>
#include "environment.hpp"
#include "main.hpp"
```

Include dependency graph for agent.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class Agent
- class ModelUndo

Enumerations

- enum update_t { action_update, percept_update }

6.2.1 Enumeration Type Documentation

6.2.1.1 enum update_t

Enumerator:

action_update

percept_update

6.3 coinflip.cpp File Reference

```
#include <cassert>
#include "coinflip.hpp"
```

Include dependency graph for coinflip.cpp:

6.4 coinflip.hpp File Reference

```
#include "environment.hpp"
```

Include dependency graph for coinflip.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class CoinFlip

6.5 environment.cpp File Reference

```
#include <cassert>
#include <sstream>
#include "environment.hpp"
#include "util.hpp"
```

Include dependency graph for environment.cpp:

6.6 environment.hpp File Reference

```
#include <string>
#include "main.hpp"
#include "util.hpp"
```

Include dependency graph for environment.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class Environment

6.7 extendedtiger.cpp File Reference

```
#include <cassert>
#include "extendedtiger.hpp"
#include "util.hpp"
```

Include dependency graph for extendedtiger.cpp:

6.8 extendedtiger.hpp File Reference

```
#include "environment.hpp"
#include "main.hpp"
```

Include dependency graph for extendedtiger.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class ExtendedTiger

6.9 kuhnpoker.cpp File Reference

```
#include <cassert>
#include "kuhnpoker.hpp"
#include "util.hpp"
```

Include dependency graph for kuhnpoker.cpp:

6.10 kuhnpoker.hpp File Reference

```
#include <string>
#include "environment.hpp"
```

Include dependency graph for kuhnpoker.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class KuhnPoker

6.11 main.cpp File Reference

```
#include <cassert>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <iostream>
#include <string>
#include <sstream>
#include "agent.hpp"
#include "environment.hpp"
#include "main.hpp"
#include "search.hpp"
#include "util.hpp"
#include "coinflip.hpp"
#include "kuhnpoker.hpp"
#include "maze.hpp"
#include "relay-maze.hpp"
#include "pacman.hpp"
#include "rock-paper-scissors.hpp"
#include "tictactoe.hpp"
#include "tiger.hpp"
#include "extendedtiger.hpp"
```

Include dependency graph for main.cpp:

Functions

- int main (int argc, char ∗argv[ ])
- void mainLoop (Agent &ai, Environment &env, options_t &options)
- void processOptions (std::ifstream &in, options_t &options)

Variables

- std::ofstream logger

6.11.1 Function Documentation

6.11.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Entry point of the program. Sets up logging, default configuration values, environment and agent before starting the agent/environment interaction cycle by calling mainLoop(). In the case of invalid command

line arguments, it prints help information to the standard output and exits.

6.11.1.2 void mainLoop (Agent & *ai,* Environment & *env,* options_t & *options*)

The main agent/environment interaction loop. Each interaction cycle begins with the agent receiving an observation and reward from the environment. Subsequently, the agent selects an action and informs the environment. The interactions that took place are logged to the logger and compactLogger streams. When the cycle equals a power of two, a summary of the interactions is printed to the standard output.

Parameters

ai The agent.

env The environment.

options The configuration options.

6.11.1.3 void processOptions (std::ifstream & *in,* options_t & *options*)

Parse configuration options from a stream. Each line of the stream should be a key/value pair of the form "key=value". Whitespace and anything following a comment "#" character are ignored.

Parameters

in The stream from which to read the configuration options.

options The map which will populated with the options.

6.11.2 Variable Documentation

6.11.2.1 std::ofstream logger

A log of the agents interactions with the environment is written to this stream in comma-separated-value format.

6.12 main.hpp File Reference

```
#include <fstream>
#include <map>
#include <string>
#include <vector>
```

Include dependency graph for main.hpp: This graph shows which files directly or indirectly include this file:

Typedefs

- typedef interaction_t action_t
- typedef long long age_t
- typedef int interaction_t

- typedef std::map< std::string, std::string > options_t
- typedef interaction_t percept_t
- typedef double reward_t
- typedef std::vector< symbol_t > symbol_list_t
- typedef bool symbol_t

Variables

- std::ofstream logger

6.12.1 Typedef Documentation

6.12.1.1 typedef interaction_t action_t

Describes an agent action.

6.12.1.2 typedef long long age_t

Describes the age of an agent.

6.12.1.3 typedef int interaction_t

Describes an interaction (observation, reward, or action) between the agent and the environment.

6.12.1.4 typedef std::map<std::string, std::string> options_t

The program's keyword/value option pairs.

6.12.1.5 typedef interaction_t percept_t

Describes a percept (observation or reward).

6.12.1.6 typedef double reward_t

Describes the reward accumulated by an agent.

6.12.1.7 typedef std::vector<symbol_t> symbol_list_t

A list of symbols.

6.12.1.8 typedef bool symbol_t

The symbols that can be predicted.

6.12.2 Variable Documentation

6.12.2.1 std::ofstream logger

A log of the agents interactions with the environment is written to this stream in comma-separated-value format.

6.13 mainpage.txt File Reference

6.14 maze.cpp File Reference

```
#include <cassert>
#include <cstdlib>
#include <iostream>
#include <limits>
#include <sstream>
#include "maze.hpp"
```

Include dependency graph for maze.cpp:

6.15 maze.hpp File Reference

```
#include "environment.hpp"
```

Include dependency graph for maze.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class Maze

6.16 pacman.cpp File Reference

```
#include <cassert>
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <sstream>
#include "pacman.hpp"
#include "util.hpp"
```

Include dependency graph for pacman.cpp:

6.17 pacman.hpp File Reference

```
#include <string>
#include "environment.hpp"
```

Include dependency graph for pacman.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class PacMan

6.18 predict.cpp File Reference

```
#include <cassert>
#include <cmath>
#include "predict.hpp"
#include "util.hpp"
```

Include dependency graph for predict.cpp:

Variables

- static const double log_half = std::log(0.5)

6.18.1 Variable Documentation

6.18.1.1 const double log_half = std::log(0.5) [static]

The value $\ln(0.5)$. This value is used often in computations and so is made a constant for efficiency reasons.

6.19 predict.hpp File Reference

```
#include <vector>
#include "main.hpp"
```

Include dependency graph for predict.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class ContextTree
- class CTNode

Typedefs

- typedef int count_t
- typedef double weight_t

6.19.1 Typedef Documentation

6.19.1.1 typedef int count_t

Stores symbol occurrence counts.

6.19.1.2 typedef double weight_t

Holds context weights.

6.20 relay-maze.cpp File Reference

```
#include <cassert>
#include <cstdlib>
#include <iostream>
#include <limits>
#include <sstream>
#include "relay-maze.hpp"
```

Include dependency graph for relay-maze.cpp:

6.21 relay-maze.hpp File Reference

```
#include "environment.hpp"
```

Include dependency graph for relay-maze.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class RelayMaze

6.22 rock-paper-scissors.cpp File Reference

```
#include <cassert>
#include <sstream>
#include "rock-paper-scissors.hpp"
#include "util.hpp"
```

Include dependency graph for rock-paper-scissors.cpp:

6.23 rock-paper-scissors.hpp File Reference

```
#include "environment.hpp"
```

Include dependency graph for rock-paper-scissors.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class RockPaperScissors

6.24 search.cpp File Reference

```
#include <cassert>
#include <cmath>
#include <limits>
#include "agent.hpp"
#include "search.hpp"
#include "util.hpp"
```

Include dependency graph for search.cpp:

Variables

- static const double exploration_constant = 2.0

6.24.1 Variable Documentation

6.24.1.1 const double exploration_constant = 2.0 [static]

Exploration constant for UCB action policy.

6.25 search.hpp File Reference

```
#include "main.hpp"
```

Include dependency graph for search.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class SearchNode

Typedefs

- typedef std::map< interaction_t, SearchNode * > child_map_t
- typedef long long visits_t

Enumerations

- enum nodetype_t { chance, decision }

6.25.1 Typedef Documentation

6.25.1.1 typedef std::map<interaction_t, SearchNode*> child_map_t

Mapping used to store the children of a SearchNode. Indexed by actions or percepts as appropriate.

6.25.1.2 typedef long long visits_t

Type for storing the number of visits to a node.

6.25.2 Enumeration Type Documentation

6.25.2.1 enum nodetype_t

Used to specify the type of SearchNode. Chance nodes represent a set of possible observation (one child per observation) while decision nodes represent sets of possible actions (one child per action). Decision and chance nodes alternate.

Enumerator:

chance

decision

6.26 tictactoe.cpp File Reference

```
#include <cassert>
#include "tictactoe.hpp"
#include "util.hpp"
```

Include dependency graph for tictactoe.cpp:

6.27 tictactoe.hpp File Reference

```
#include "environment.hpp"
```

Include dependency graph for tictactoe.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class TicTacToe

6.28 tiger.cpp File Reference

```
#include <cassert>
#include <sstream>
#include "util.hpp"
#include "tiger.hpp"
```

Include dependency graph for tiger.cpp:

6.29 tiger.hpp File Reference

```
#include "environment.hpp"
```

Include dependency graph for tiger.hpp: This graph shows which files directly or indirectly include this file:

Data Structures

- class Tiger

6.30 util.cpp File Reference

```
#include <cassert>
#include <cstdlib>
#include <iostream>
#include "util.hpp"
```

Include dependency graph for util.cpp:

Functions

- int bitsRequired (const int x)
- interaction_t decode (const symbol_list_t &symbols, const int bits)
- void encode (symbol_list_t &symbols, interaction_t value, const int bits)
- double rand01 ()
- int randRange (int start, int end)
- int randRange (int end)
- void requiredOption (options_t const &options, const std::string option_name)

6.30.1 Function Documentation

6.30.1.1 int bitsRequired (const int *x*)

Calculate the number of bits needed to store x >= 0.

6.30.1.2 interaction_t decode (symbol_list_t const & *symbols*, const int *bits*)

Decodes a specified number of bits from the end of a symbol list.

Parameters

symlist The list of symbols to decode from.

bits The number of bits from the end of the symbol list to decode.

Returns

The decoded value as an integer.

6.30.1.3 void encode (symbol_list_t & *symbols*, interaction_t *value*, const int *bits*)

Encode a value onto the end of a list of symbols using a specified number of bits.

Parameters

symlist The list onto which to encode the value.

value The value to be encoded.

bits The number of bits of the value to encode onto the symbol list.

6.30.1.4 double rand01 ( )

Sample a number from the unit interval uniformly at random.

Returns

A random double between 0 and 1.

6.30.1.5 int randRange (int *start*, int *end*)

Sample an integer from a specified range uniformly at random.

Parameters

start The start of the range (inclusive) to sample from.

end The end of the range (exclusive) to sample from.

Returns

A random integer greater than or equal to start and less than end.

6.30.1.6 int randRange (int *end*)

Sample an integer from a specified range uniformly at random.

Parameters

end The end of the range (exclusive) to sample from.

Returns

A random integer greater than or equal to 0 and less than end.

6.30.1.7 void requiredOption (options_t const & *options*, const std::string *option_name*)

Checks if an option is present in the options map. If not, the program exits.

Parameters

options The options map.
option_name The option to search for.

6.31 util.hpp File Reference

```
#include <sstream>
#include <string>
#include "main.hpp"
```

Include dependency graph for util.hpp: This graph shows which files directly or indirectly include this file:

Functions

- int bitsRequired (const int x)
- interaction_t decode (symbol_list_t const &symbols, const int bits)
- void encode (symbol_list_t &symbols, interaction_t value, const int bits)
- template<typename T >
  T fromString (std::string const &str)
- template<typename T >
  void fromString (std::string const &str, T &value)
- template<typename T >
  T getOption (options_t &options, const std::string option_name, const T default_value)
- template<typename T >
  void getOption (options_t &options, const std::string option_name, const T default_value, T &value)
- template<typename T >
  T getRequiredOption (options_t const &options, const std::string option_name)
- template<typename T >
  void getRequiredOption (options_t const &options, const std::string option_name, T &val)
- double rand01 ()
- int randRange (int start, int end)
- int randRange (int end)
- void requiredOption (options_t const &options, const std::string option_name)
- template<typename T >
  std::string toString (const T val)

6.31.1 Function Documentation

6.31.1.1 int bitsRequired (const int *x*)

Calculate the number of bits needed to store x >= 0.

6.31.1.2 interaction_t decode (symbol_list_t const & *symbols,* const int *bits*)

Decodes a specified number of bits from the end of a symbol list.

Parameters

symlist The list of symbols to decode from.

bits The number of bits from the end of the symbol list to decode.

Returns

The decoded value as an integer.

6.31.1.3 void encode (symbol_list_t & *symbols,* interaction_t *value,* const int *bits*)

Encode a value onto the end of a list of symbols using a specified number of bits.

Parameters

symlist The list onto which to encode the value.

value The value to be encoded.

bits The number of bits of the value to encode onto the symbol list.

6.31.1.4 template<typename T > T fromString (std::string const & *str*)

Extract a value of type T from a string.

Parameters

str The string from which to extract the value.

Returns

The extracted value.

6.31.1.5 template<typename T > void fromString (std::string const & *str,* T & *value*)

Extract a value of type T (e.g. an integer) from a string.

Parameters

str The string from which to extract the value.

value The variable into which to extract the value.

6.31.1.6 template<typename T > T getOption (options_t & *options*, const std::string *option_name*, const T *default_value*)

Get the value of a non-required option.

Parameters

options The options map.

option_name The option to retrieve.

default_value The default value of the option (if not present).

Returns

The retrieved value.

6.31.1.7 template<typename T > void getOption (options_t & *options*, const std::string *option_name*, const T *default_value*, T & *value*)

Get the value of a non-required option.

Parameters

options The options map.

option_name The option to retrieve.

default_value The default value of the option (if not present).

value Stores the retrieved value.

6.31.1.8 template<typename T > T getRequiredOption (options_t const & *options*, const std::string *option_name*)

Get the value of a required option. If the option is not present in the options map the program exits.

Parameters

options The options map.

opt The option to retrieve.

Returns

The value of the option.

6.31.1.9 template<typename T > void getRequiredOption (options_t const & *options*, const std::string *option_name*, T & *val*)

Get the value of a required option. If the option is not present in the options map the program exits.

Parameters

options The options map.

option_name The option to retrieve.

val Stores the retrieved value.

6.31.1.10 double rand01 ( )

Sample a number from the unit interval uniformly at random.

Returns

A random double between 0 and 1.

6.31.1.11 int randRange (int *start*, int *end*)

Sample an integer from a specified range uniformly at random.

Parameters

start The start of the range (inclusive) to sample from.

end The end of the range (exclusive) to sample from.

Returns

A random integer greater than or equal to start and less than end.

6.31.1.12 int randRange (int *end*)

Sample an integer from a specified range uniformly at random.

Parameters

end The end of the range (exclusive) to sample from.

Returns

A random integer greater than or equal to 0 and less than end.

6.31.1.13 void requiredOption (options_t const & *options*, const std::string *option_name*)

Checks if an option is present in the options map. If not, the program exits.

Parameters

options The options map.

option_name The option to search for.

6.31.1.14 template<typename T > std::string toString (const T *val*)

Convert a value into a string.