Desarrollo de un programa de computador capaz de jugar Starcraft®: Brood War™ usando técnicas de aprendizaje por refuerzo

Diego A. Ballesteros Villamizar

Universidad de los Andes

October 27, 2011

Outline

Outline

Motivation

- During the development of this project several scenarios were explored, each with a different model
- But the algorithms remained the same, then some code was copy-pasted among projects
- So it is convenient to create a library that can be adapted to different problems and implements the reinforcement learning algorithms

Similar Work

- A framework for reinforcement learning is not something new
- RL-Glue is a software that provides a language-independent interface to connect different RL software blocks [1]
- RL-Toolbox is a C++ library that implements several algorithms and a extensive framework that includes the agent, controller and environment. Has a special focus on optimal control of continuous tasks [2]

[1] Tanner, B., & White, A. (2009). RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, *10*, 2133–2136

[2] Neumann, G. (2005). *The Reinforcement Learning Toolbox, Reinforcement Learning for Optimal Control Tasks*. Master's thesis, Technische Universität Graz, Austria

So Why Implement It?

- RL-Toolbox is a great option and it is in C++ but not updated since 2006
- To gain better insight on reinforcement learning techniques I consider it is better to implement them
- Also due to time constraints it is a safer path that understanding and extending another framework
- Setup a start point that can be extended by other people interested in reinforcement learning, a simpler yet useful RL framework for education

Outline

1. RL Library
   - Introduction
   - Design
   - Implementation
   - Tests
2. Scenario with incomplete game information
   - A Quick Recap
   - Introduction
   - Implementation and Results
3. Test Case #3
   - Introduction
   - Implementation
4. To Do List

A View of the Reinforcement Learning Framework

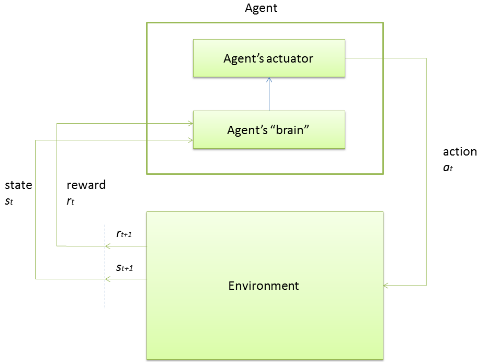


Figure: Depiction of the proposed RL Framework

Basic Reinforcement Learning Process

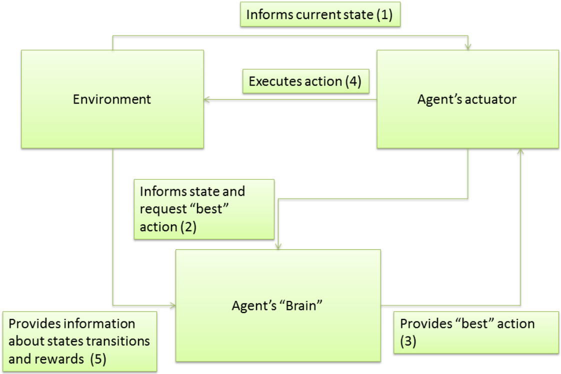


Figure: Illustration of the basic RL process, the numbers indicate the actions' order

Library Design
Considerations

- The library will represent the agent's "brain"
- Must process the environment information, implements a learning algorithm and indicates the actions to take in every state
- Should define a general interface so it can be integrated with multiple problems, represented by an environment (states) and an actuator (actions) as well as state transitions (environment's dynamics) and a reward model

Library Design
Scope

- For now only discrete reinforcement learning problems are considered
- Only model free TD(0) algorithms will be implemented (Q-Learning and SARSA)
- Most of the work consists in the implementation of a general framework for discrete problems

Figure: Trying to avoid the scope creep

Library Design
Problem Representation

- Problems can be represented as a collection of states and actions in every state (state-action pairs)
- States can be represented as sets of multi-valued features otherwise states must have an unique label
- Actions can be represented with an unique label
- Use of states represented by features can help fast search of state-action pairs, by indexing them using the features' values

Library Design

Algorithms' Data Representation

- Once a problem has been represented in terms of its state-action pairs then learning information must be stored too
- For Q-Learning and SARSA the only needed information are Q-values
- A lexicographical order of state-action pairs is performed and with this they are indexed for faster search
- Q-Values can be stored in an array indexed the same way as state-action pairs
- Features can be used to build indexes and have instant access to the information

Library Design
Information Storage

- Problems can be stored in text files and loaded to the library during execution
- Q-Values can be stored in text files, and loaded posteriorly to continue the learning
- Resulting policies can also be generated and stored in a text file
- These files' format will be shown shortly

Outline

Class Diagram
Problem Representation

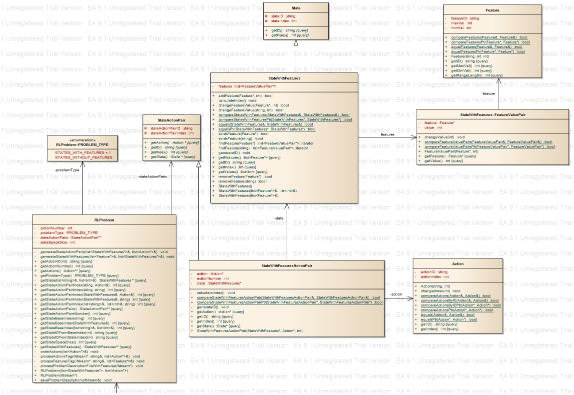


Figure: Classes used to describe a RL problem

Class Diagram

Algorithms' implementation

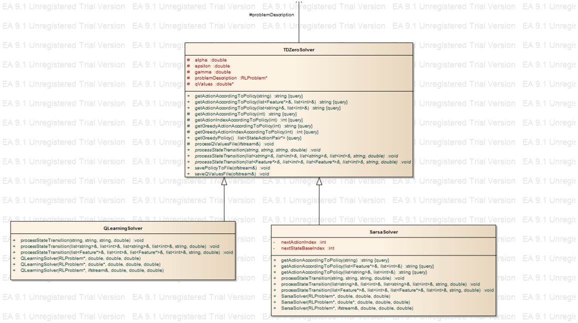


Figure: Classes that implement the reinforcement learning algorithms

Problem Description File

```
#Lines started with '#' are comments lines and are ignored
#The first line to be processed must be the type line
#Therefore every line must be either a comment line or
#a line with proper formatting
#For the formats following in the comments, a SSSS represents a string
# and a ### represents a positive integer
#The type line has the following format
#Type=###
#1 represents a problem defined with states, defined by features, and actions
#2 represents a problem with states, without features, and actions
Type=1
#After type comes the problem description
#The next non-comment line must be a tag
#Possible tags are:
#[Features] only if type was set to 1
#[States] only if type was set to 2
#[Actions] for any type
[Features]
#Each feature is described in a single line, the description must be written
#in the following format
#ID=SSSS;Min=###;Max=###
ID=X;Min=0;Max=4
ID=Y;Min=0;Max=4
#After all items are described the next non-comment line must be one
#of the remaining tags depending on the type of problem description selected
#Tags must not be repeated
[Actions]
#Following an actions tag comes the definition of all available actions
#Each definition must be in the following format
#Action=SSSS
Action=UP
Action=DOWN
Action=LEFT
Action=RIGHT
```

Figure: General file's format

```
#This file describes the states and actions for the windyGridWorld
Type=1
[Features]
ID=X-Coordinate;Min=0;Max=9
ID=Y-Coordinate;Min=0;Max=6
[Actions]
Action=UP
Action=DOWN
Action=LEFT
Action=RIGHT
```

Figure: An example of a problem description file

Q-Values File

```
#A comment line is preceded by a '#' symbol
#In every automatically generated file the first line
#will be the following comment line:
#"This file has been automatically generated"
#Next there will be a line indicating the type o
#f information contained in the file, in this case:
#"This file contains learned Q-values"
#Next comes the type of problem, that is either 1 or 2
Type=#
#The next line is a tag indicating the start of
#the Q-Values information
[Q-Values]
#Then each line will contain a state ID
#separated by a ';' from the action and
#then another ';' before the associated Q-Value
STATE 1 ; MOVE ; Q-Value= 0.32
STATE 1 ; DANCE ; Q-Value= 0.40
```

Figure: General file's format

```
#This file has been automatically generated
#This file contains learned Q-values
Type=1
[Q-Values]
X-Coordinate= 0 ; Y-Coordinate= 0 ; DOWN ; Q-Value= -6.26433
X-Coordinate= 0 ; Y-Coordinate= 0 ; LEFT ; Q-Value= -6.28287
X-Coordinate= 0 ; Y-Coordinate= 0 ; RIGHT ; Q-Value= -6.19639
X-Coordinate= 0 ; Y-Coordinate= 0 ; UP ; Q-Value= -6.20517
X-Coordinate= 0 ; Y-Coordinate= 1 ; DOWN ; Q-Value= -6.57774
X-Coordinate= 0 ; Y-Coordinate= 1 ; LEFT ; Q-Value= -6.55397
X-Coordinate= 0 ; Y-Coordinate= 1 ; RIGHT ; Q-Value= -6.54256
X-Coordinate= 0 ; Y-Coordinate= 1 ; UP ; Q-Value= -6.57537
X-Coordinate= 0 ; Y-Coordinate= 2 ; DOWN ; Q-Value= -7.13986
X-Coordinate= 0 ; Y-Coordinate= 2 ; LEFT ; Q-Value= -7.266
X-Coordinate= 0 ; Y-Coordinate= 2 ; RIGHT ; Q-Value= -7.13619
X-Coordinate= 0 ; Y-Coordinate= 2 ; UP ; Q-Value= -7.11861
X-Coordinate= 0 ; Y-Coordinate= 3 ; DOWN ; Q-Value= -7.82849
X-Coordinate= 0 ; Y-Coordinate= 3 ; LEFT ; Q-Value= -7.93717
X-Coordinate= 0 ; Y-Coordinate= 3 ; RIGHT ; Q-Value= -7.84577
X-Coordinate= 0 ; Y-Coordinate= 3 ; UP ; Q-Value= -7.82417
X-Coordinate= 0 ; Y-Coordinate= 4 ; DOWN ; Q-Value= -7.77527
X-Coordinate= 0 ; Y-Coordinate= 4 ; LEFT ; Q-Value= -7.85447
X-Coordinate= 0 ; Y-Coordinate= 4 ; RIGHT ; Q-Value= -7.78909
X-Coordinate= 0 ; Y-Coordinate= 4 ; UP ; Q-Value= -7.82938
```

Figure: An example of a Q-Values file

Policy File

```
#A comment line is preceded by a '#' symbol
#In every automatically generated file the first line
#will be the following comment line:
#"This file has been automatically generated"
#The next line is a tag indicating the start of
#the policy information
[Policy]
#Then each line will contain a state ID
#separated by a ';' from the best action found after
#learning
STATE 1 ; DANCE
STATE 2 ; MOVE
```

Figure: General file's format

```
#This file has been automatically generated
[Policy]
X-Coordinate= 0 ; Y-Coordinate= 0 ; RIGHT
X-Coordinate= 0 ; Y-Coordinate= 1 ; RIGHT
X-Coordinate= 0 ; Y-Coordinate= 2 ; UP
X-Coordinate= 0 ; Y-Coordinate= 3 ; UP
X-Coordinate= 0 ; Y-Coordinate= 4 ; DOWN
X-Coordinate= 0 ; Y-Coordinate= 5 ; UP
X-Coordinate= 0 ; Y-Coordinate= 6 ; LEFT
X-Coordinate= 1 ; Y-Coordinate= 0 ; UP
X-Coordinate= 1 ; Y-Coordinate= 1 ; LEFT
X-Coordinate= 1 ; Y-Coordinate= 2 ; UP
X-Coordinate= 1 ; Y-Coordinate= 3 ; DOWN
X-Coordinate= 1 ; Y-Coordinate= 4 ; DOWN
X-Coordinate= 1 ; Y-Coordinate= 5 ; DOWN
X-Coordinate= 1 ; Y-Coordinate= 6 ; RIGHT
X-Coordinate= 2 ; Y-Coordinate= 0 ; DOWN
X-Coordinate= 2 ; Y-Coordinate= 1 ; RIGHT
X-Coordinate= 2 ; Y-Coordinate= 2 ; UP
X-Coordinate= 2 ; Y-Coordinate= 3 ; DOWN
X-Coordinate= 2 ; Y-Coordinate= 4 ; RIGHT
X-Coordinate= 2 ; Y-Coordinate= 5 ; LEFT
X-Coordinate= 2 ; Y-Coordinate= 6 ; LEFT
X-Coordinate= 3 ; Y-Coordinate= 0 ; RIGHT
X-Coordinate= 3 ; Y-Coordinate= 1 ; RIGHT
```

Figure: An example of a policy file

Outline

Testing on a Windy Gridworld

Windy Gridworld Problem

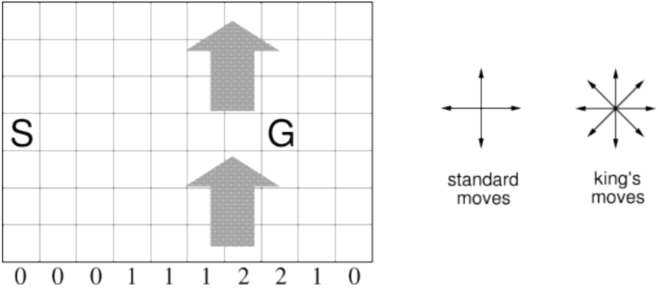


Figure: Image of the windy GridWorld problem [3]

[3] Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning : an introduction*. A Bradford Book. The MIT Press

Testing on a Windy Gridworld
Comparison with Sutton & Barto's Book

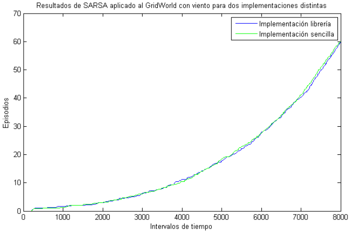


Figure: Obtained results using the implemented library

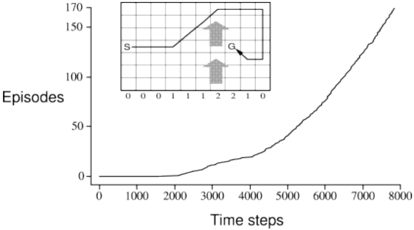


Figure: Reinforcement Learning book's results

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning : an introduction*. A Bradford Book. The MIT Press

Testing on a Windy Gridworld

Comparison and effect of learning rate

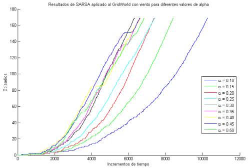


Figure: Obtained results using the implemented library

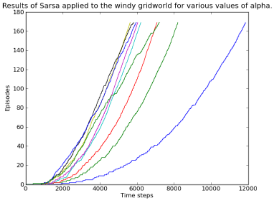


Figure: Results obtained by a student at Stellenbosch University

Schaaf, H. (2011). The basics of reinforcement learning. URL `http://ml.sun.ac.za/2010/06/18/the-basics-of-reinforcement-learning/`

Testing on the Cliff-Walking Problem

Cliff Walking Problem

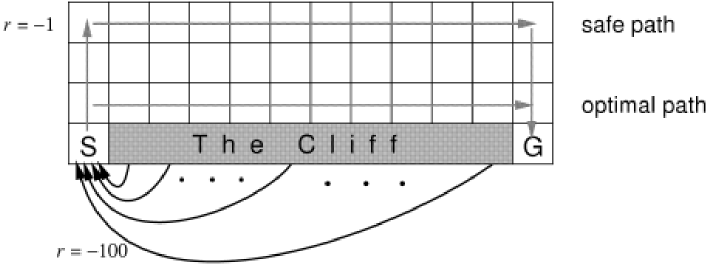


Figure: Depiction of the Cliff-Walking problem [4]

[4] Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning : an introduction*. A Bradford Book. The MIT Press

Testing on the Cliff-Walking Problem

Comparing Q-Learning and SARSA

Figure: Obtained results using the implemented library

Figure: Reinforcement Learning book's results

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning : an introduction*. A Bradford Book. The MIT Press

Outline

Map Details

- Open map, no obstacles
- 128x128 Tiles (4096x4096 pixels)

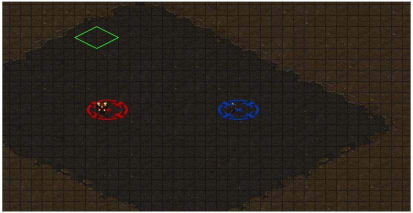


Figure: Map editor view

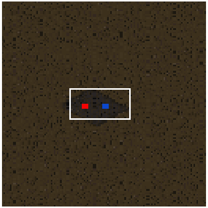


Figure: Minimap view

Unit Details

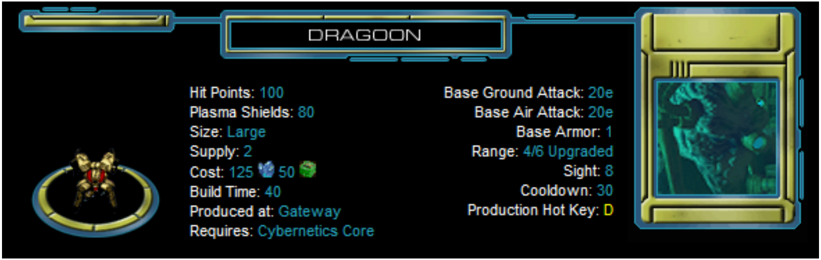


Figure: Dragoon unit information

Unit Details

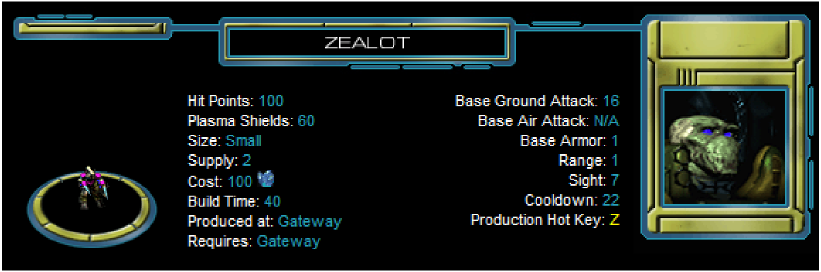


Figure: Zealot unit information

Unit Details

- Bot: 1 Dragoon
- Enemy: 1 Zealot
- A Zealot can do 0.72 Damage per second against a Dragoon
- A Dragoon can do 0.33 Damage per second against a Zealot
- 12 Hits from a Zealot kill a Dragoon, that would take 264 frames (ca. 11 seconds)
- 16 Hits from a Dragoon kill a Zealot, that would take 480 frames (ca. 20 seconds)
- The zealot has the upper hand

Unit Details

- Bot: 1 Dragoon
- Enemy: 1 Zealot
- A Zealot can do 0.72 Damage per second against a Dragoon
- A Dragoon can do 0.33 Damage per second against a Zealot
- 12 Hits from a Zealot kill a Dragoon, that would take 264 frames (ca. 11 seconds)
- 16 Hits from a Dragoon kill a Zealot, that would take 480 frames (ca. 20 seconds)
- Maybe Not?

Outline

What changes?

- So far the reinforcement learning agent has cheated!
- A complete map information flag was enabled and the agent knew every detail of the game
- So how does the agent perform if some information is hidden?

Motivation

- Real world problems do not necessarily have all the information available at all times
- In Starcraft most information about the enemy is usually hidden to the player
- Fog of War hides units' position and status
- Implementing working algorithms in this environment is important

Outline

How to deal with the new condition?

- Partial Observable Markov Decision Processes describe this kind of problem
- There literature describes algorithms but these are rather complicated
- For this game it may be possible to predict the states based on memory and knowledge of the game

How to deal with the new condition?

- In this particular scenario it may not be relevant because when the enemy is hidden it doesn't pose a threat
- The enemy always look for the Dragoon so he can just sit and wait until the environment is observable again
- When the enemy is out of sight, the Dragoon moves randomly around its position

Learning results

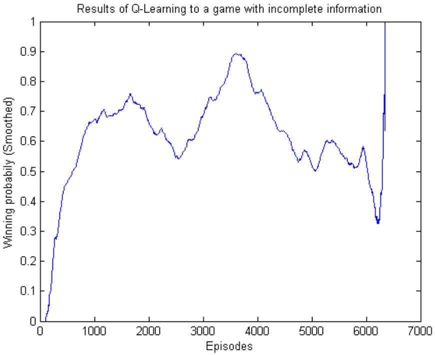


Figure: Winning probability smoothed using a moving average filter

Analysis

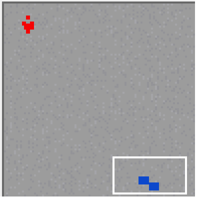
- Results are similar to those of the complete information case
- As the enemy always pursues the agent, the enemy is visible most of the time
- It could be interesting to explore the capabilities of using specfic knowledge of the game to deal with partial information in a reinforcement learning problem

Outline

Motivation

- The hardest of all experiments so far, multiple units on both sides and a completely balanced match
- Developing a good behavior that can help the player attack a position is a step in the right direction to the development of AIs using reinforcement learning
- There are countless possibilites and it is interesting to see what reinforcement learning can produce
- This experiment is also a good scenario to implement a conventional FSM-based AI for comparison

Test case's map



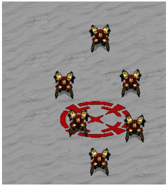


Figure: Agent's Units

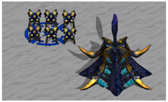


Figure: Enemy's Units

Units' Details

- 6 Dragoons for each player
- The agent's dragoons will be in one corner of the map
- The enemy's dragoons will remain in the opposite corner and will be defending the zone
- The game is balanced and micromanagement decides the outcome

Unit's Details

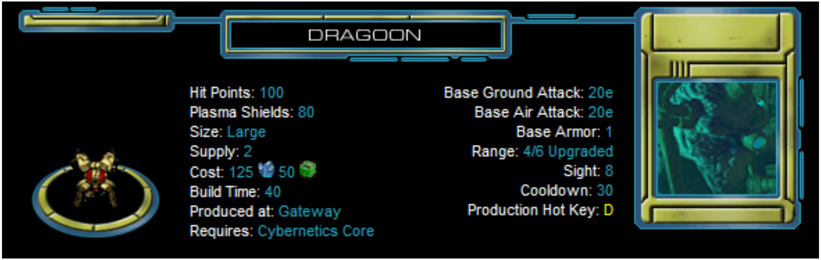


Figure: Dragoon unit information

Objective

- Control the group of dragoons and move them to the objective and destroy any enemies encountered

Outline

Actions

Reinforcement Learning Setup

- Move to Target
- Retreat from Target
- Get in some formation
- Attack weakest unit in range
- Attack weakest unit in group range
- Attack weakest unit in sight
- Attack weakest unit in group sight
- Retreat units under attack
- Retreat weakest units under attack

Formations
Actions

- Form a straight line: To cover more area and attack multiple enemies at once
- Form an arc: To focus fire in tightly grouped enemies
- Group: To move near a point before moving
- Disperse: Break formation to explore more territory

States

- Formation Status: Forming, In Formation (Which one?)
- Units Alive: In, discretized, %
- Enemies Killed: In, discretized, %
- Distance to Target
- Enemies in Group Range: In, discretized, %

Schedule

- Get results from third test case, November 13
- Prepare final presentation and document, Week of November 14
- Hand final document to supervisor, November 21
- Hand final document to reviewers, November 30
- Presentation, December 7 or 9