

Report for Assessment 1

of CS3616

Joana Welte, Philipp Lucas

14th April 2011

Contents

1	Scenario	3
2	Reinforcement Learning	3
2.1	States	3
2.2	Actions	3
2.3	Graphs	3
3	How to play	3

1 Scenario

In our scenario there is one fast ogre (let's call him John) in fraction "blue" and a variable number of slower creatures in fraction "yellow". The primary goal of the fast ogre is to escape from the dungeon by reaching the exit. For this it is not necessary to kill all the enemies however it might become necessary to fight against some of them as they block the way out.

The normal weapon used by John is the longbow, as this allows him to attack without being attacked himself. In addition he can decide to pickup use potions to regain health or energy.

2 Reinforcement Learning

The learning techniques we decided to employ is reinforcement learning.

2.1 States

We use the following state variables, all of which are divided into 5 equidistant intervals:

- health points of Tom
- energy points of Tom
- distance between Tom and closest enemy

2.2 Actions

Tom can take the following actions:

- attack closest enemy
- evade close enemies
- escape to exit
- get and use closest health potion
- get and use closest energy potion

"closest" in this context means absolute distance and "not distance to walk".

"evade close enemies" lets Tom move directly away from the geometric center of all enemies that are within a certain radius from Tom.

2.3 Implementation

We implemented the reinforcement learning in a rather generic way.

Qtable.java This class holds a Q-table to be used in reinforcement learning. As long as Action.java and State.java provide all necessary methods Q-Table is entirely independent on the actual actions or states. It provides a number of methods to update the qtable and retrieve actions:

static void updateTable (double reward, State oldState, State newState, Action oldAction) updates the Q-table using the given parameters, i.e. in our case the previous state John was in, the current state John is in, the action that led to this state and the reward that is to be assigned for this.

public static Action getRandomAction() returns a random action, i.e. one of those defined in Action.java.

public static Action getGreedyAction(State state) returns the greediness % the best, otherwise a random action.

There are 4 classes needed:

Action.java Action.java is an enumeration class of all actions that can be taken.

2.4 Graphs

3 How to play