Task 3. Reinforcement Learning

Garoe Dorta-Perez
CM50245: Computer Animation and Games II

March 19, 2015

1 Introduction

2 A* path finding

The A* algorithm can find paths optimally in a graph based on a heuristic. It works under the assumption that the distance heuristic will never overestimate the cost of the path and the path cost will not decrease as we travel through it. In a maze world, where the movement possibilities are north, south, east and west, with fixed positive costs. Using a Manhattan distance for the actual path cost and a euclidean distance to compute the heuristic will fit the assumptions.

In Algorithm 1 an overview of a general implementation of the A* in pseudocode is given. As stated in the previous paragraph, in our case, graph.cost() and heuristic() returns respectively, the Manhattan and the euclidean distance between two nodes. An example output with a small labyrinth where a path is successfully found is shown in Figure 1.

3 TD learning in noughts and crosses

Temporal-difference is a reinforcement learning technique used when a problem can be modelled as a Markov decision process, the world is modelled as a chain of *states*, where an agent can take *actions*, each of which are associated with a new state and a *reward*. The objective is to maximize the rewards that the agents gets, and by doing so taking the optimal actions in that given world. However, since the system is an online learner, there is a trade-off between *exploitation*, where the agent takes the actions that lead to the highest expected reward, and *exploration*, where the agent takes different actions to gain more information of unexplored states.

Each state s has a value $V(s)$ associated to it; the agent will choose either, a greedy *exploitation* action by performing an action a such that the next state s' will be $\max V(s_{t+1})$, or a *exploration* action a such that the next state s' will chosen randomly. When an end state is reached, the values for each state where a greedy action was taken will be updated by a backtracking mechanism as $V(s) = V(s) + \alpha\left[V(s') - V(s)\right]$, where α is the learning rate. In order to have a policy update which asymptotically guarantees converge, it is a necessary condition for α to decrease its value over time.

Algorithm 1: A*

Data: *goal* goal position, *start* start position, *graph* graph with the tiles in the map.
Result: *path* path from *start* to *goal*, *cost* total cost of the path.

```
frontier = PriorityQueue(start, 0);
came_from[start] = None;
cost_so_far[start] = 0;
while not frontier.empty() do
    current = frontier.get();
    if current == goal then
        break;
    end
    for next in graph.neighbors(current) do
        new_cost = cost_so_far[current] + graph.cost(current, next);
        if next not in cost_so_far or new_cost <cost_so_far[next] then
            cost_so_far[next] = new_cost;
            priority = new_cost + heuristic(goal, next);
            frontier.put(next, priority);
            came_from[next] = current;
        end
    end
end
path = getPath(came_from);
goal = cost_so_far[current];
```

Figure 1: Path finding example, S is the start position, G is the goal position, $*$ are the path points, . are empty spaces and # are walls.

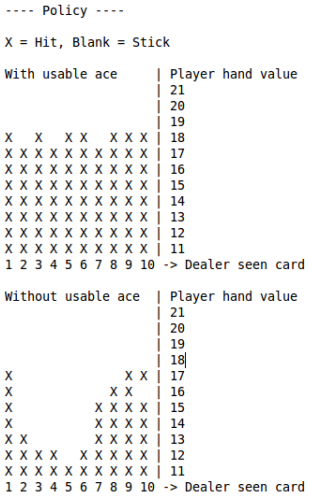

Figure 2: Blackjack policy example, with 20% exploration rate after 100000 games.

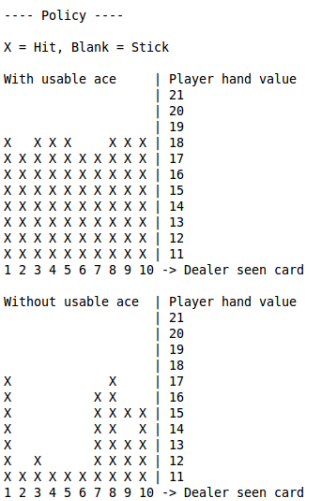

Figure 3: Blackjack policy example, with 40% exploration rate after 100000 games.

Nought and crosses can be modelled as a Markov decision process given that a board state characterizes the sum of the moves that lead to such state. The *agent* is one of the players, a *state* is a given configuration of a board and the *reward* is given when a game ends. The agent will receive a high positive reward when it wins, a negative reward when it looses and a small positive reward when drawing.

In the implementation the state space will be modelled in a tree as shown in Figure 4. On the root of the tree, we have empty board, the children are the 9 possible plays for the crosses player, who will always be the first to play.

To test our system, the agent would play a fixed number of games against an AI with fixed moves initialized at random. In Figures 6, 7 and 8 each curve represents the value for placing a cross in each of the 9 available positions as the first move of a new game. The reward for winning, drawing and loosing a game was set to 10, 1 and -1 respectively. Since the adversary of the agent is deterministic, a first move that leads to a winning combination is quickly discovered and its value increased. In Figure 5 a comparison of mean gains after playing a fixed number of games is shown; higher *exploitation* rates entail increased mean rewards, as expected in a simple game with a deterministic opponent.

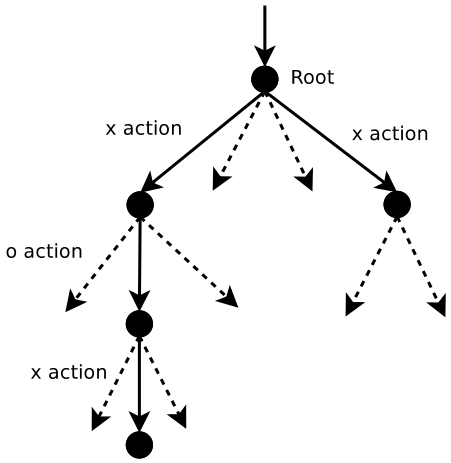

Figure 4: Noughts and crosses tree state.

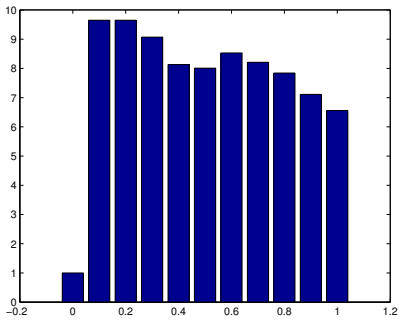

Figure 5: Mean gain after playing 100000 games of noughts and crosses with the given exploration rates.

4 Q-learning in BlackJack

For Q-learning we retain the assumptions made in Section 3. However, the state structure will be a lookup table Q instead of a tree. Q is a matrix of size $m \times n$, where m is the number of states and n are the number of actions. Every time the agent takes an action a, the Q value of the pair $\{s, a\}$ is updated as follows $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$, where α is the learning rate, r_{t+1} is the reward for being in s_{t+1} and γ is a discounting factor. An important advantage of using the aforementioned update mechanism is its off-policy nature, since the $Q(s_t, a_t)$ will be updated with

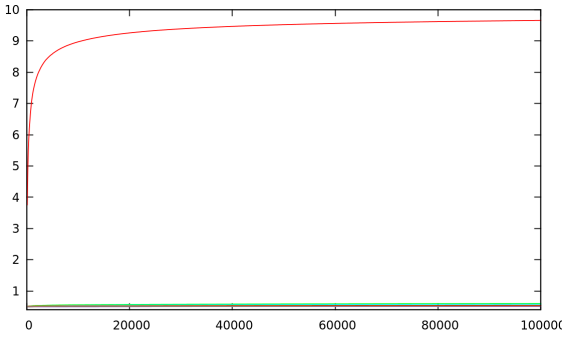

Figure 6: Values for the first movement for noughts and crosses with 20% exploration rate after 100000 games.

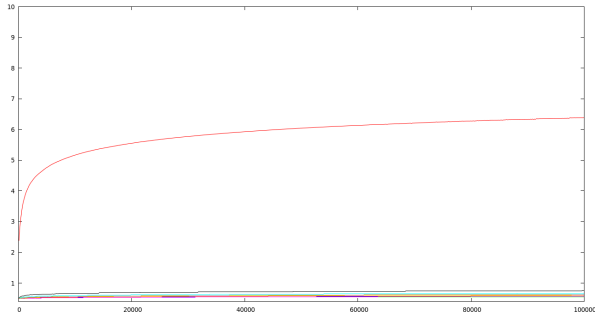

Figure 7: Values for the first movement for noughts and crosses with 40% exploration rate after 100000 games.

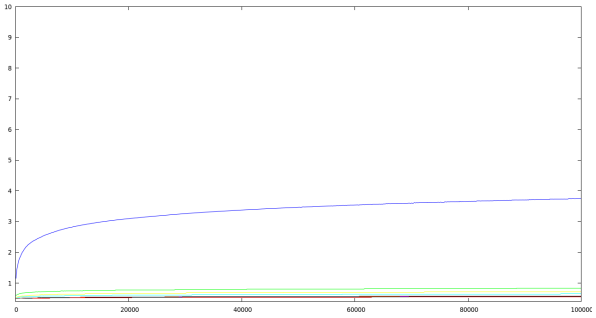

Figure 8: Values for the first movement for noughts and crosses with 60% exploration rate after 100000 games.

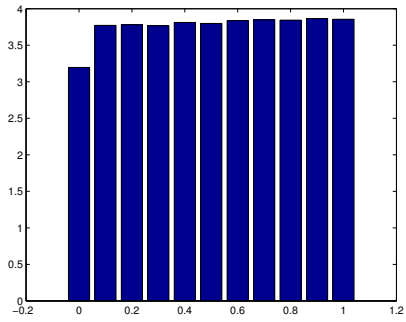

Figure 9: Expected gain after playing 100000 games of blackjack with the given exploration rates.

$\max_a Q(s_{t+1}, a)$, even if the policy chooses an action which is suboptimal, the update will be computed with the optimal $Q(s_{t+1}, a)$ value.

Our blackjack implementation adopts the following rules:

- A 52 french card deck is used.
- The player is dealt an initial two cards hand.
- The player may count one ace as 1 or 11 points.
- The dealer must hit cards until they have 17 or more points.
- If the dealer and the player have the same number of points, the dealer wins.

With a reward of 10 for winning and -1 for loosing the mean gain for different exploration rates is shown in Figure 9. Since the dealer has an slight advantage, using equal rewards for winning and loosing produces a negative expected gains, as shown in Figure 10.

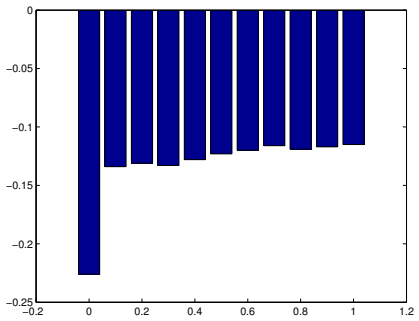

Figure 10: Expected gain after playing 100000 games of blackjack with the given exploration rates.

5 Results and conclusions

We have presented basic algorithms for path finding and reinforcement learning that are commonly uses in games. With A* we have an method to will output the optimal path with a minimal node evaluation. The heuristic distance could be overestimated in order to have a suboptimal path with early termination. More advance path finding methods include D* for incremental maps or Theta* for a continuous space map. A substantial advantage of Q-learning is its online learning component, therefore if implement as the AI of the game, it can adapt to changes in the player strategy trough its continuous learning process.