Task 3. Reinforcement Learning

Garoe Dorta-Perez
CM50245: Computer Animation and Games II

March 20, 2015

1 Introduction

2 A* path finding

The A*[1] algorithm can find the shortest paths in a graph, searching within a minimal number of nodes for a given 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 the Manhattan distance for the real path cost, and the euclidean distance to compute the heuristic cost will fit the aforementioned assumptions.

In Algorithm 1, an overview of a pseudocode implementation of the A* is shown. As stated in the previous paragraph, graph.cost() and heuristic() returns the Manhattan and the euclidean distance between two nodes, respectively. The key idea, is that promising directions, those with a lower heuristic cost will be explored first, thus the exploration favours directions that move towards the goal. Moreover, since the heuristic will always underestimate the cost, if a path looks good from an heuristic point of view, yet it is actually poor choice, when the path is explored, its cost will lower its priority in the frontier, so the search will continue at the next best guess. 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 learning[4] 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 the given world. However, since the system is an online learner, there is a trade-off between *exploitation*, where the agent takes actions that lead to the highest expected reward, and *exploration*, where the agent takes suboptimal actions to acquire information from unexplored states.

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];
```

```
 X 00000000001111111111
   01234567890123456789
Y
00 S*******************
01 .###################*
02 .##..###.#.#.#.###**
03 .##..###.#.#.#.###*.
04 .#....##.#.#****##*.
05 .#..#....#***.#****.
06 .####.....*####.....
07 .########.*.########.
08 .#........G#........
09 .#.#######..#######.
10 .#....#..#..........
11 .#####.#.#.#####....
12 .#.#.###.#.#.#.###..
13 .#.#.###.#.#.#.###..
14 .#....##.#.#....##..
15 .#..#....#....#.....
16 .####......####.....
17 ..#######...###.####
18 .#.........#........
19 ......................
```

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.

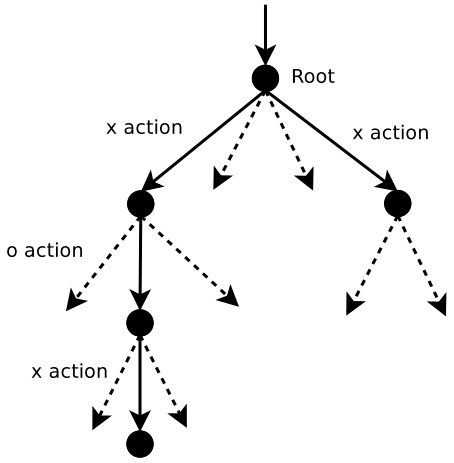

Figure 2: Noughts and crosses tree state.

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 be 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 $V(s) = V(s) + \alpha\left[V(s') - V(s)\right]$, where α is the learning rate. In order to have a policy update which asymptotically guarantees convergence, it is a necessary condition for α to decrease its value over time.

Nought and crosses can be modelled as a Markov decision process, given that each 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 rewards are delivered when a game ends. The agent will receive a high positive reward when they win, a negative reward when they loose and a small positive reward for drawing.

In the implementation, the state space will be modelled in a tree as shown in Figure 2. 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. Its children the next possible moves for the noughts player and so on.

To test our system, the agent would play a fixed number of games against an AI with fixed moves initialized randomly. In Figures 3, 4 and 5, each curve represents the value for placing a cross in each of the 9 available positions in a new game as the first move. 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 is therefore increased. In Figure 6, 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.

4 Q-learning in BlackJack

For Q-learning[4], 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 is the number of actions. Every time the agent takes an action a_t, the Q value of the pair $\{s_t, a_t\}$ 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 the state 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 $\max_a Q(s_{t+1}, a)$, even if the policy chooses an action which is suboptimal, the update will be computed with the optimal $Q(s_{t+1}, a)$ value.

Lets define the rules used for our blackjack implementation:

- Cards are drawn from a 52 cards french deck.
- Cards points are the numeric value shown on the card, however kings, queens and jacks value is 10.
- The player may count one ace with 1 or 11 points.

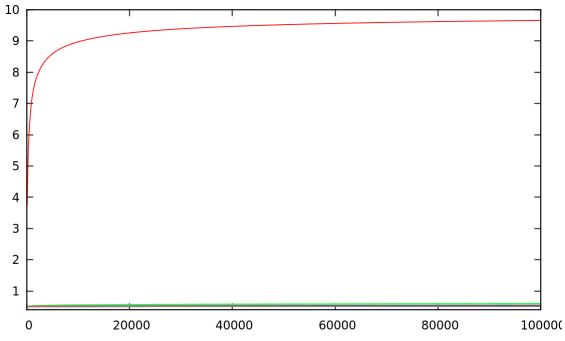

Figure 3: Values for the first movement for noughts and crosses with 20% exploration rate after 100000 games.

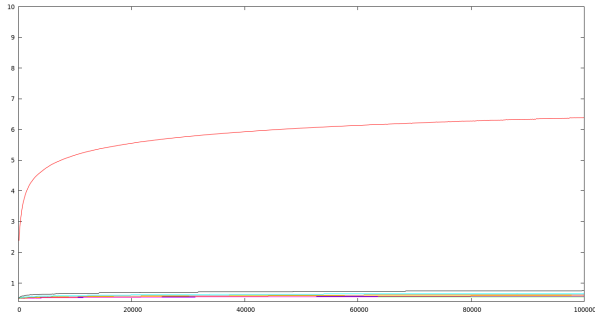

Figure 4: Values for the first movement for noughts and crosses with 40% exploration rate after 100000 games.

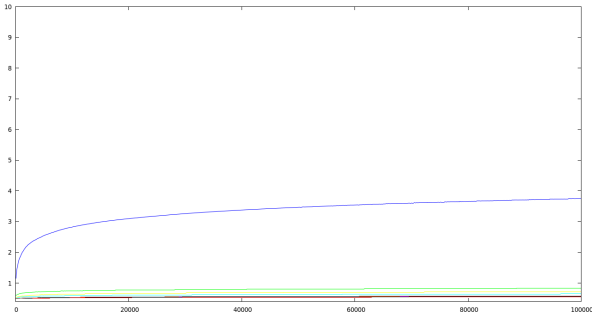

Figure 5: Values for the first movement for noughts and crosses with 60% exploration rate after 100000 games.

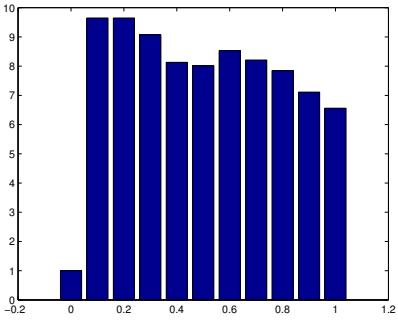

Figure 6: Mean gain after playing 100000 games of noughts and crosses with the given exploration rates.

- The player is dealt an initial two cards hand.
- 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 negative expected gains, as shown in Figure 10. Policies learned after playing 100000 games are shown in Figures 7 and 8. With an ace in the hand, in both policies the agent stops hitting cards after their value is close to 18. The agent plays more cautiously if it does not have any ace. This policy makes sense since an ace can be counted as 11 points, so if the agent goes over 21 it can be recounted as 1 point. Comparing both policies, we can see that with a higher exploration rate, the agent learnt that it should play even more prudently without an ace.

```
---- Policy ----

X = Hit, Blank = Stick

With usable ace     | Player hand value
                    | 21
                    | 20
                    | 19
X   X   X X   X X X | 18
X X X X X X X X X X | 17
X X X X X X X X X X | 16
X X X X X X X X X X | 15
X X X X X X X X X X | 14
X X X X X X X X X X | 13
X X X X X X X X X X | 12
X X X X X X X X X X | 11
1 2 3 4 5 6 7 8 9 10 -> Dealer seen card

Without usable ace  | Player hand value
                    | 21
                    | 20
                    | 19
                    | 18
X               X X | 17
X             X X   | 16
X           X X X X | 15
X           X X X X | 14
X X         X X X X | 13
X X X X   X X X X X | 12
X X X X X X X X X X | 11
1 2 3 4 5 6 7 8 9 10 -> Dealer seen card
```

Figure 7: Blackjack policy example, with 20% exploration rate after 100000 games.

```
---- Policy ----

X = Hit, Blank = Stick

With usable ace     | Player hand value
                    | 21
                    | 20
                    | 19
X   X X X     X X X | 18
X X X X X X X X X X | 17
X X X X X X X X X X | 16
X X X X X X X X X X | 15
X X X X X X X X X X | 14
X X X X X X X X X X | 13
X X X X X X X X X X | 12
X X X X X X X X X X | 11
1 2 3 4 5 6 7 8 9 10 -> Dealer seen card

Without usable ace  | Player hand value
                    | 21
                    | 20
                    | 19
                    | 18
X               X   | 17
X             X X   | 16
X           X X X X | 15
X           X X   X | 14
X           X X X X | 13
X   X       X X X X | 12
X X X X X X X X X X | 11
1 2 3 4 5 6 7 8 9 10 -> Dealer seen card
```

Figure 8: Blackjack policy example, with 40% exploration rate after 100000 games.

5 Results and conclusions

We have presented basic algorithms for path finding and reinforcement learning, which can be applied in games. With A* we have a method that will output the optimal path, with minimal node evaluation for a given heuristic. The heuristic distance could be overestimated in order to have a suboptimal path with early termination. More advance path finding methods include D*[3] for incremental maps or Theta*[2] for a continuous space map. A substantial advantage of Q-learning, is its online learning component, therefore if implemented as the

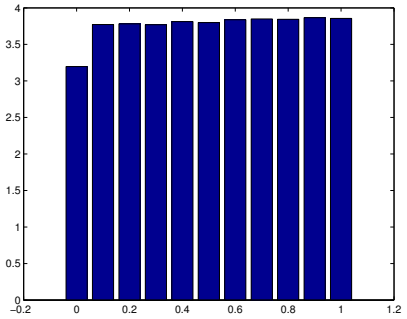

Figure 9: Expected gain after playing 100000 games of blackjack with the given exploration rates.

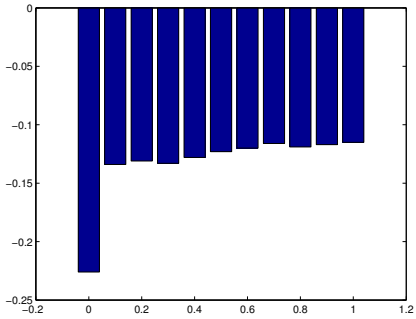

Figure 10: Expected gain after playing 100000 games of blackjack with the given exploration rates.

AI of the game, it can adapt to changes in the player strategy trough its continuous learning process.

References

[1] P.E. Hart, N.J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[2] Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. Theta*: Any-Angle Path Planning on Grids. In *Proceedings of the National Conference on Artificial Intelligence*, page 1177, 2007.

[3] Anthony Stentz and Is Carnegle Mellon. Optimal and Efficient Path Planning for Partially-Known Environments. In *International Journal of Robotics and Automation*, volume 10, pages 89–100, 1993.

[4] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.