Mid-Term-Report Blackjack-Agent

Environment

For this project we are using python. We have made a blackjack simulator based on an interactive blackjack application that we found on GitHub. Currently it simulates an agent playing against the bank with a regular deck size (four copies of each card). We will do further experimentation with the deck size and playing against multiple opponents.

Algorithms

We have already implemented some deterministic algorithms to have a benchmark for comparison (Hit until bust, Bank strategy, and Expected value strategy).

```
def naive_action(player_hand, dealer_hand):
  """
  Hit until bust.
  :param player_hand: list of cards
  :param dealer_hand: single card
  :return: (action, empty)
  """
  return "h", ()
```

```
def hit_until_17(player_hand, dealer_hand):
  """
  Hit until score > 17 or bust
  :param player_hand: list of cards
  :param dealer_hand: single card
  :return: (action, empty)
  """
  action = "h"
  if bj.total(player_hand) >= 17:
    action = "s"
  return action, ()
```

```
def expected_value_strategy(player_hand, dealer_hand):
  """
  Hit if (score + (expected value of next cared)) <= 21
  :param player_hand: list of cards
  :param dealer_hand: single card
  :return: (action, empty)
  """
  number_of_remaining_cards = (52*bj.number_of_decks) - len(player_hand) -
1
  total_A1 = (340*bj.number_of_decks)
  ace = False
  for card in player_hand:
    if card == "A":
        ace = True
  if not(ace):
```

```
    if not(dealer_hand == "A"):
        expected_value_1 = (total_A1 - bj.total(player_hand) -
bj.total(str(dealer_hand))) / number_of_remaining_cards
    else:
        expected_value_1 = (total_A1 - bj.total(player_hand) - 1) /
number_of_remaining_cards
    if bj.total(player_hand) + expected_value_1 <= 21:
        return ("h", ())
    else:
        return ("s", ())
  else:
    total_player, used_aces = bj.total(player_hand, True)
    if used_aces == 1:
        if not(dealer_hand == "A"):
            expected_value_1 = (total_A1 + 10 - bj.total(player_hand) -
bj.total(str(dealer_hand))) / number_of_remaining_cards
        else:
            expected_value_1 = (total_A1 + 10 - bj.total(player_hand) - 1)
/ number_of_remaining_cards
        # Compare cases like: (A + 9 = 20 -> stand) and (A + 5 = 16 -> hit)
        if bj.total(player_hand) + expected_value_1 <= 21 or
(bj.total(player_hand) + expected_value_1 - 10 <= 21 and
bj.total(player_hand) + expected_value_1 - 10 > bj.total(player_hand)) or
(bj.total(player_hand) + expected_value_1 - 10 <= 21 and
bj.total(player_hand) + 2*expected_value_1 - 10 > bj.total(player_hand) and
bj.total(player_hand) + 2*expected_value_1 - 10 <= 21):
            return ("h", ())
        else:
            return ("s", ())
    else:
        if not(dealer_hand == "A"):
            expected_value_1 = (total_A1 - bj.total(player_hand) -
bj.total(str(dealer_hand))) / number_of_remaining_cards
        else:
            expected_value_1 = (total_A1 - bj.total(player_hand) - 1) /
number_of_remaining_cards
        if bj.total(player_hand) + expected_value_1 <= 21:
            return ("h", ())
        else:
            return ("s", ())
```

Hit until bust:

Will only succeed if a “blackjack” is obtained.

Bank strategy:

Hit until a total of 17 points is reached, then stand independent of the banks card.

Expected value strategy:

Hit if score + expected value of next card <= 21. This takes into consideration the cards known to the agent.

Experimental Results

We did our first tests with 1’000’000 games against the bank with a single full deck (4 copies per card) and got the following expected returns:

-Hit until bust: -0.72 * bet

-Bank strategy: -0.079 * bet

-Expected value strategy: -0.062 * bet

We then tested the expected value strategy in 1’000’000 games with 6 decks and got an expected return of -0.065 * bet. Since the knowledge is low compared to the deck size in both cases, we expected a small decrease in the expected return.

Plan for remainder of the project

As we get further along in the lectures, we will see which reinforcement-learning algorithms fit best for our purpose and implement those. If possible, we will also adapt and test our strategies for games with multiple players. This would lead to more information and a greater chance to get a positive expected return.