```
// A program that implements a simple simulation of Poker for 4 players.
// Only recognizes single pairs, three-of-a-kinds, and four-of-a-kinds.

Include
{
        "stdlib/stdlib.cgl";
}

CardEntities
{
        dealer;
        player0;
        player1;
        player2;
        player3;
}

Globals
{
        var players;
}

Start // Deal cards, set chips
{
        var i;
        var e;

        << "Hello and Welcome to PCGSL Poker!";

        #players = [$player0, $player1, $player2, $player3];

        << "Shuffling deck.";
        shuffle($dealer);
        << "Dealing Cards.";
        i = 0;
        while (i < |#players|) {
              e = #players[i];

              // Deal five cards to the player.
              e <- $dealer[0];
              e <- $dealer[0];
              e <- $dealer[0];
              e <- $dealer[0];
              e <- $dealer[0];

              // Print out the player's hand.
              << "Player " ^ i ^ "'s hand: " ^ e[0] ^ " " ^ e[1] ^ " " ^ e[2]
                 ^ " " ^ e[3] ^ " " ^ e[4];

              i++;
        }
}

Play
{
}

WinCondition
{
        var i;
        var comp;
        var highplayers;
        var highhand;
        var best0;
        var best1;
        var best2;
        var best3;

        best0 = best_hand($player0);
```

```
        << "Player 0 best hand : " ^ best0[0] ^ " of face " ^ best0[1];
        best1 = best_hand($player1);
        << "Player 1 best hand : " ^ best1[0] ^ " of face " ^ best1[1];
        best2 = best_hand($player2);
        << "Player 2 best hand : " ^ best2[0] ^ " of face " ^ best2[1];
        best3 = best_hand($player3);
        << "Player 3 best hand : " ^ best3[0] ^ " of face " ^ best3[1];

        comp = hand_compare(best0, best1);
        if (comp > 0) {
           highplayers = [$player0];
           highhand = best0;
        } else {
           if (comp < 0) {
              highplayers = [$player1];
              highhand = best1;
           } else {
              highplayers = [$player1, $player2];
              highhand = best0;
           }
        }

        comp = hand_compare(highhand, best2);
        if (comp < 0) {
           highplayers = [$player2];
           highhand = best2;
        } else {
           if (comp == 0) {
              highplayers = highplayers :: $player2;
           }
        }

        comp = hand_compare(highhand, best3);
        if (comp < 0) {
           highplayers = [$player3];
           highhand = best3;
        } else {
           if (comp == 0) {
              highplayers = highplayers :: $player3;
           }
        }

        << "The winners are:";
        i = 0;
        while (i < |highplayers|) {
              << highplayers[i];
              i++;
        }

        return highplayers;
}


// Returns [type, val] where type is an int for the type of hand (4 = four
// of a kind, 3 = 3 of a kind, 2 = best pair, 1 = high card), and val is the
// value of that type of hand (14 = Aces, 13 = Kings, ..., 2 = 2's). Assumes
// a 5-card CardEntity is given.
best_hand(var e) {
        var list;
        var i;
        var highpairval;
        var highsingval;

        list = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];

        list[cardface(e[0])] += 1;
        list[cardface(e[1])] += 1;
        list[cardface(e[2])] += 1;
        list[cardface(e[3])] += 1;
```

```
        list[cardface(e[4])] += 1;

        i = 2;
        while (i <= 14) {
           if (list[i] == 4) {
              return [4, i];
           }

           if (list[i] == 3) {
              return [3, i];
           }

           if (list[i] == 2) {
              highpairval = i;
           }

           if (list[i] == 1) {
              highsingval = i;
           }

           i++;
        }

        if (highpairval != null) {
           return [2, highpairval];
        }

        return [1, highsingval];
}

// Takes two hands of the form [type, val] as described in the high_hand
// function. Returns 1 if the first hand is better, -1 if the second hand is
// better, and 0 if they are equal.
hand_compare(var h1, var h2) {

        if (h1[0] > h2[0]) {
           return 1;
        }

        if (h2[0] > h1[0]) {
           return 0 - 1;
        }

        if (h1[1] > h2[1]) {
           return 1;
        }

        if (h2[1] > h1[1]) {
           return 0 - 1;
        }

        return 0;
}
```