ISEL / ADEETC

Informatics and Multimedia Engineering

**Multimedia Authoring**

# 2nd Laboratory Work

# Multimedia Authoring

## Blackjack – cards game

Rui Jesus e João Beleza

## Introduction

This work aims to introduce and familiarize with the JavaScript programming language. The goal is to develop the Blackjack card game. Each student / group has to implement the methods / functions that are to be done in the code provided by the teacher. Students are free to make changes as long as they do not stray too far from the structure.

Below is a link to the w3schools website that should be consulted during game development.

http://www.w3schools.com/

## Objectives

The project provided by the teacher includes 3 files: "blackjack_oop.html", "blackjack_manager.js" and "blackjack_object.js". The first implements the user interface ("View"), the second implements the interface between the user interface and the "blackjack" ("Controller") class and the third contains the class code (Model). This implementation follows the MVC (Model, View, Controller) design standard that allows the application to be decomposed into several components, more suitable for code maintenance and reuse. Figure 1 illustrates the MVC design pattern.

At the end of the first lesson the student must have at least the blackjack class implemented, that is, the student should make the object code of the methods. At the end of the second lesson the student should have the application implemented, that is, the functions of the file "blackjack_manager.js". The full Blackjack game code has to be delivered by October 29, 2017 using the Moodle.

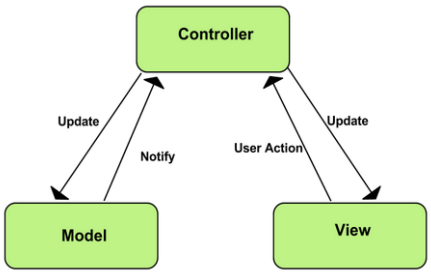


Figure 1. MVC Components.

## Blackjack

**1. Objective:** add 21 points with the cards in hand.

**2. Participating elements:** the dealer who gives the cards and the player who plays against the dealer.

**3. Card score:** cards 2 through 10 punctuate with the value marked on the card (2 to 10 points); the King, the Lady and the Jack are worth 10 points and the Aces can have a value of 1 or 11 (the player chooses how it suits him).

**4. Initial conditions:** Two cards are dealt to dealer and one for the player. The dealer's 2nd card faces down (the card is not visible).

**5. Game Dynamics:** After the first card, the player can move the game to the dealer or ask for more cards. If the player's card score exceeds 21 points, the player loses immediately. If the score is 21, the player wins immediately. If the player passes the game to the dealer, the dealer plays until he has 21 points or exceeds the player's score. If the dealer exceeds the player's score, he wins, if he breaks, loses.

**6. End of the game:** (1) player makes 21 points; (2) player bursts; (3) dealer makes more points than the player; (4) dealer bursts.

## Laboratory work

### "blackjack" class

1. Unzip the "blackjack.zip" file and create a new project in IntelliJ IDEA with the files provided (Make "NEW-> Project from Existing Sources").

2. Implement the following methods of the "blackjack" class:
   a. `new_deck()`

      This method returns an array with the 52 cards represented by numbers from 1 to 13 for each suit. Numbers 11, 12 and 13 represent the figures (King, Jack and Lady).

   b. `shuffle()`

      This method shuffles the array of cards built in the previous method and returns a new shuffled array.

      You should create an array of indexes from 1 to 52 (for). Then you must do another "for" and for each cycle (52) to draw (Math.random) an index. This index is used to fetch a card from the deck. This card is inserted at the end of

another array with the shuffled cards. Do not forget to remove the drawn index from the index array.

c. `get_cards_value()`

This method counts the value of an array of cards (dealer_cards or player_cards) according to the rules of blackjack. The cards from 2 to 10 punctuate with the value marked on the card (2 to 10 points). The King, the Lady and the Jack are worth 10 points and the Aces can have a value of 1 or 11 (the player chooses how it suits him). Returns the resulting points.

d. `get_game_state()`

This method checks whether the dealer's and player's card scores allow you to finish the game (bust or 21) and if someone wins. Updates the variable "state" (member of the Blackjack class) with the current state and returns this variable.

e. `player_move()`

This method will fetch the next card from the deck and place it in the player's card array. Returns the updated "state" variable by executing the `get_game_state()` method that updates the game state after the player rolls.

f. `dealer_move()`

This method is the same as before but now for the dealer (the new card must be placed in the Dealer's card array.

3. Implement the functions of the file "blackjack_manager.js":

a. `new_game()`

This function creates an instance of the class "blackJack", executes two moves of the dealer and one of the player. Do not forget that the dealer's 2nd card must appear face down. For example, you must replace the dealer's 2nd line display with the "X" character. Finally, the user interface buttons are initialized.

b. `update_dealer()`

This function checks if the game state is "gameEnded". If so, ask the "blackjack" class for the dealer's cards. Check if the dealer won or lost. And start building a string to show the cards by adding information to the string if the dealer won or

lost. Finally, it updates the string in the HTML element associated with the dealer and executes the finalize_buttons () function.

c. `update_player()`

This function asks the class "blackJack" the player's cards and starts building a string to show the cards. Next, it checks to see if the game status is "gameEnded" and if the player has won/loose and it adds information to the string if the user won or lost. Then, it updates the string in the HTML element associated with the player and executes the finalize_buttons () function.

d. `dealer_new_card()`

This function executes the "blackJack" class method that performs the dealer's move, updates the dealer, and returns the state of the game.

e. `player_new_card()`

This function executes the "blackJack" class method that performs the player move, updates the player and returns the state of the game.

f. `dealer_acaba()`

This function calls the get_game_state () method of the "blackJack" class and sets "true" to the "DealerTurn" variable of the "blackJack" class. Then a loop is created until the game finishes (state.gameEnded). In this cycle, the dealer is updated, a dealer's move is performed and the game's state is updated at each iteration.

4. Improve the user interface (optional).
5. Add rules used in some casinos that are not included in this simplified version (optional).