Team **Ghost Dragons**

Console Game Application **Dragonjack**

Team members:

# Павел Ангелов (PawelAngelow)

# Гергана Белчева (gbelcheva)

# Емилия Георгиева (Emiliya93)

# Красимир Димитров (gua543)

# Надежда Величкова-Кънева (NadiaKaneva)

# Михаил Нейков (mihailn)

# Цвятко Павлов (cvqtko.pavlov)

# Стоян Парасков (Stoyan001)

# Dragonjack

# Dragonjack is a game based on the popular card game Blackjack. The main objective is to achieve Blackjack by getting 2 cards with a sum of 21. Alternatively, you can win by having a bigger sum than the dealer. If the total sum of the cards becomes more than 21 or less than the dealer’s, you lose. The dealer plays by set casino rules – hits until 17 and stands on all 17s or more.

# The game title screen consists of an ASCII art animation that presents the game title. The next screen simulates a Blackjack table with a shuffled deck of cards. During gameplay the application displays card sum, graphical card faces in player and dealer hands and a menu with player options. The user interacts with keystrokes. The game allows the player to place a bet and collect wins. Dragonjack also features sound effects and a player score screen that compares the player achievement with the top 10 players of all time.

# The game is played in the following manner:

# After the initial animation of the title screen, the user is prompted to continue by pressing ENTER. On the next screen, the player starts with an initial capital of $1000 and is allowed to bet any amount from their funds. The next stage is the actual game. The player plays with the keys Z, X, C and V according to a menu of available options. At first, the player can *double down* which means they double their bet and are dealt one more card before their turn ends. If the player’s first two cards are matching with the same value, they can split their hand into two hands each with their own bet and play them separately. When player ends their turn, the dealer plays in compliance with casino rules (stands on all 17s). After everyone ends their turns and the outcome of the play is clear, the bet is calculated like this: if the player has a Dragonjack (first two cards equal 21), the dealer pays their bet 3:2; if the player scores more than the dealer and less than 22, the dealer pays their bet 1:1; if the dealer matches the player cards, no one wins and the player collects their original bet; if the player scores less than the dealer or more than 21, or the dealer has Dragonjack, the dealer collects the bet. The point system of different cards is the following: 2, 3, 4, 5, 6, 7, 8, 9, 10 correspond to their numbers; J, Q, K have a value of 10; A can have a value of 1 or 11 as the player decides. At this point, the player collects their wins and may continue their game by pressing ENTER or leave the table and cash in their winnings by pressing F12. If the player chooses to continue playing, the table is cleared, the user is prompted to place another bet, and the dealer deals cards for another play. If the player chooses to leave the game or is out of money, the game continues and may prompt for player’s initials, if the player has achieved a top 10 score. Then, the game displays the top 10 players of all time with their results. At any point the user can exit the console application by pressing ESC.

# General Requirements:

# At least 1 multi-dimension array – Used for storing deck/decks of cards and for ASCII art.

# At least 3 one-dimensional arrays – Arrays are used for storing the player and dealer hands, at least two arrays are used for visualising card faces.

**At least 10 methods (separating the application’s logic)** – The game logic is separated in methods (more than 10), some of them in their own classes.

**At least 3 existing .NET classes** – Some of the .NET classes used are: *System.Random* class to simulate a shuffled deck of cards, *System.Console* class to set the console window colour and size, and *System.ConsoleKey* for user interaction.

**At least 2 exception handlings** – The application catches an *ArgumentOutOfRange* exception when setting the console window and buffer sizes and displays a helpful message for the user. Also, when reading and writing to the external text file and using media files, it catches exceptions if the file is missing, or corrupted, etc. Another exception handling is used when the player places their bet.

**At least 1 use of external text file** – The game keeps track of best players by saving their results in a text file.

Optional requirements:

**OOP** – The application uses classes with methods and constructors for code readability and to improve cohesion and lower coupling.

**Sound effects** during gameplay.

Git repository at GitHub – <https://github.com/GhostDragons/Dragon-Jack>

# Credits:

# Casino sounds by Kenney Vleugels ([www.kenney.nl](http://www.kenney.nl))

# [www.high-impact-sound-effects.com](http://www.high-impact-sound-effects.com)