CMSC 131 Introduction to Computer Organization & Machine-level Programming
CAPSTONE PROJECT

BLACK JACK

By:

[BS CS III] CHRISTIAN JEREMY M, LAPUZ

[BS CS III] TIMOTHY RIVER R. PALER

TABLE OF CONTENTS

PROJECT SUMMARY

Project Description

This was made in compliance with the requirements for CMSC 131, Introduction to Computer Organization & Machine-level Programming. The project requires teams to design and develop a video game which should be unique, meaning no duplicate games among teams. The game should be coded or programmed using only the Machine-level programming language known as Assembly which was introduced during the start of the sem. Assembly is also known as a low-level programming language. Low-level programming languages, such as Assembly, have syntax structures which are as close to machine code as possible. This means there is not as much translation from syntax to machine language as compared to high-level programming languages causing compilation to be relatively faster. The project aims to test all the learnings obtained over the course of the semester, and to showcase them to people who are possibly not informed on the possibilities of the Assembly programming language.

Game Summary

The selected concept revolves around the game of Blackjack. Blackjack, also known as Twenty-One, is a comparing card game which utilizes all 52 cards in a standard deck. The game consists of a player, or players, and a dealer. The dealer is in charge of distributing the cards amongst the players and theirselves. Each card has a value which would added to the player's or dealer's current points. Cards 1 through 10's value corresponds to their number, while the cards Jack, Queen and King each have a value of 10. The Ace card can either be a value of 1 or 11, depending on the player's decision. The objective of the game is simple. In order for the player to win, they must either get points exactly summing up to 21, or get a point value which is greater than the dealer's without exceeding 21. The game starts of with each person having one face up card among them. The player can then choose either to "hit" or to "stand". When hit is chosen, they

are given a card by the dealer. When stand is chosen, the player is no longer given any cards, and the dealer starts drawing cards for themselves until their points reaches or is over 17. A combination of an Ace card and a 10 card is known as a natural or “blackjack”, since the Ace can have a value of 11 totaling to 21 points with only two cards. Any other two card combination with an Ace is known as a “soft hand” since the Ace can either have a value of 1 or 11, depending on the player.

GitHub Link

https://github.com/CjLapuz/BlackJack

PROCEDURES

Main Procedure

- Contains the main logic for the game and the flow of procedure calls and input listeners. Also in charge of outputting the right displays

```
MAIN PROC FAR

 MOV AX, @data
 MOV DS, AX
 MOV ES, AX

 ;---- M A I N  M E N U  S T A T E S -------
 MOV AH, 3DH
 MOV AL, 00
 LEA DX, START1_PATH        ; open start is selected display
 INT 21H
 MOV FILEHANDLE, AX
 MOV AH, 3FH
 MOV BX, FILEHANDLE
 MOV CX, 2000
 LEA DX, START_1            ; save file contents to a variable
 INT 21H
 MOV AH, 3EH              ;request close file
 MOV BX, FILEHANDLE
 INT 21H

 MOV AH, 3DH
 MOV AL, 00
 LEA DX, START2_PATH        ; open how to play is selected display
 INT 21H
 MOV FILEHANDLE, AX
 MOV AH, 3FH
 MOV BX, FILEHANDLE
 MOV CX, 2000
 LEA DX, START_2            ; save file contents to a variable
 INT 21H
 MOV AH, 3EH              ;request close file
 MOV BX, FILEHANDLE
 INT 21H

 MOV AH, 3DH
 MOV AL, 00
 LEA DX, START3_PATH        ; open exit is selected display
 INT 21H
 MOV FILEHANDLE, AX
```

```
  MOV AH, 3FH
  MOV BX, FILEHANDLE
  MOV CX, 2000
  LEA DX, START_3            ; save file contents to a variable
  INT 21H
  MOV AH, 3EH               ;request close file
  MOV BX, FILEHANDLE
  INT 21H

  MOV AH, 3DH
  MOV AL, 00
  LEA DX, HOW_PATH          ; open how to play display
  INT 21H
  MOV FILEHANDLE, AX
  MOV AH, 3FH
  MOV BX, FILEHANDLE
  MOV CX, 2000
  LEA DX, HOW             ; save file contents to a variable
  INT 21H
  MOV AH, 3EH               ;request close file
  MOV BX, FILEHANDLE
  INT 21H
  ;---- E N D  O F  L O A D I N G  O F  S T A T E S ----------

START_SLCT:
  CALL _CLEAR_SCREEN
  MOV   DL, 0               ; set cursor to 0,0
  MOV   DH, 0               ; for entire screen coverage
  CALL  _SET_CURSOR

  MOV STATE, 01             ; set state to 1
  LEA DX, START_1           ; display start is selected
  CALL DISPLAY_STR
  JMP   LISTENER
HOW_SLCT:
  MOV STATE, 02             ; set state to 3
  LEA DX, START_2           ; display exit is selected
  CALL DISPLAY_STR
  JMP   LISTENER
EXIT_SLCT:
  MOV STATE, 03             ; set state to 2
  LEA DX, START_3           ; display how to play is selected
  CALL DISPLAY_STR
  JMP   LISTENER

RIGHT_MOV:                  ; right arrow navigation
  MOV   DL, 0               ; set cursor to 0,0
  MOV   DH, 0               ; for entire screen coverage
  CALL  _SET_CURSOR
  CALL _CLEAR_SCREEN        ; reset the screen before display
```

```
  CMP STATE, 01             ; moving to how to play is selected display
  JE HOW_SLCT
  CMP STATE, 02             ; moving to exit is selected display
  JE EXIT_SLCT
  JMP START_SLCT            ; when exit is selected moves back to start is selected

LEFT_MOV:                   ; left arrow navifation
  MOV   DL, 0               ; set cursor to 0,0
  MOV   DH, 0               ; for entire screen coverage
  CALL  _SET_CURSOR
  CALL _CLEAR_SCREEN        ; reset the screen before display

  CMP STATE, 01             ; jump to exit when start is the current selected option
  JE EXIT_SLCT
  CMP STATE, 02             ; moving to start is selected display
  JE START_SLCT
  JMP HOW_SLCT              ; moving to how to play is selected display

BACK_MAIN:
  CMP STATE, 04             ; check if current screen is how to play display
  JE START_SLCT
  JMP LISTENER              ; if not do nothing

HOW_DISPLAY:
  MOV STATE, 04             ; set state to 2
  LEA DX, HOW               ; display how to play the game
  CALL DISPLAY_STR
  JMP LISTENER

  LISTENER:
     MOV NEW_INPUT, '$'     ; clear input holder
     CALL _GET_KEY          ; listen for inputs

     CMP NEW_INPUT, 4DH     ; right arrow key press
     JE RIGHT_MOV

     CMP NEW_INPUT, 4BH     ; left arrow key press
     JE LEFT_MOV

     CMP NEW_INPUT, 1B      ; esc press [;!change to enter!;]
     JE EVENT

     CMP NEW_INPUT, '$'     ; any other key press
     JNE BACK_MAIN          ; going back to title screen from how to play display

     JMP LISTENER

EVENT:
  MOV   DL, 0               ; set cursor to 0,0
```

```
  MOV   DH, 0               ; for entire screen coverage
  CALL  _SET_CURSOR
  CALL _CLEAR_SCREEN        ; reset the screen before display

  CMP STATE, 01             ; start game is selected
  JE GAME
  CMP STATE, 02             ; how to play is selected
  JE HOW_DISPLAY
  JMP FIN                   ; exit game

GAME:
  CALL PLAYER_MOVE          ; player's turn

  MOV SCOREHOLDER, 0        ; reset player's points

  ADD CURR_CARDVAL, '0'     ; convert to string value of int

  LEA DX, CURR_CARDVAL      ; display current player points
  CALL DISPLAY_STR

  CALL ENEMY_MOVE           ; dealer's turn

  MOV SCOREHOLDER, 0        ; reset dealer's points

  ADD ENEMY_CARDVAL, '0'    ; convert to string value of int

  LEA DX, ENEMY_CARDVAL     ; display current dealer points
  CALL DISPLAY_STR

  MOV AL, CURR_CARDVAL      ; moving player points to AL
  CMP AL, ENEMY_CARDVAL     ; compare if player's points are higher
  JG WIN
  JL LOSE
  JE DRAW

FIN:
  CALL _TERMINATE           ; early terminate from game screen


FINISH:
  LEA DX, EXIT_STR          ; display how to exit the game
  CALL DISPLAY_STR
B10:
  MOV NEW_INPUT, '$'        ; clear input holder
  CALL _GET_KEY             ; listen for inputs

  CMP NEW_INPUT, 1B         ; esc key press
  JNE B10                   ; loop in listening for esc key press
  CALL _TERMINATE
```

```
DRAW:
  LEA DX, PRINT_DRAW          ; Draw result display
  CALL DISPLAY_STR
  JMP FINISH
WIN:
  LEA DX, PRINT_WIN           ; Player wins display
  CALL DISPLAY_STR
  JMP FINISH
LOSE:
  LEA DX, PRINT_LOSE          ; Dealer wins display
  CALL DISPLAY_STR
  JMP FINISH
```

OTHER PROCEDURES

- **Set Cursor**
  - Sets the cursor pointer for printing and proper alignment when displaying the interfaces.

CODE:

```
_SET_CURSOR PROC  NEAR
        MOV   AH, 02H
        MOV   BH, 00
        INT   10H

          RET
_SET_CURSOR ENDP
```

- **Get Key**
  - Listens in on any form of keyboard input, and returns when no key has been pressed.

CODE:

```
_GET_KEY  PROC  NEAR
        MOV   AH, 01H               ; check for input
        INT   16H
        JZ    __LEAVETHIS           ; no input returns to the caller
        MOV   AH, 00H
        INT   16H
        MOV   NEW_INPUT, AH         ; set the value of the input
        __LEAVETHIS:
                RET
_GET_KEY  ENDP
```

- **Player Move**
  - Asks the player to either hit or stand. When hit is chosen, the give card procedure is called, if the total points reach past 21 the player loses.

When stand is chosen the total player points are saved, and would then proceed to enemy move procedure.

CODE:

```
PLAYER_MOVE PROC NEAR

LEA DX, PRINT                    ; hit or stand choice
CALL DISPLAY_STR

CALL ACCEPT_STR1                 ; take the selected choice

MOV BH, 'h'                      ; set bh to 'q'
MOV BL, 's'                      ; set bl to 's'

CMP KBINPUT1, BH                 ; check if hit
JE HIT
CMP KBINPUT1, BL                 ; check if stand
JE STAND

 HIT:
   CALL GIVE_CARD                ; call give card
   CMP SCOREHOLDER, 21           ; check if over 21
   JG LOSE
  JMP PLAYER_MOVE                ; else, player's move again

STAND:                           ; if stand return value
   JMP _RET

_ret:
   MOV BL, SCOREHOLDER           ; take the current score and save to BL
   MOV CURR_CARDVAL, BL          ; save the score value to the dealers points
   RET
PLAYER_MOVE ENDP
```

- **Enemy Move**
  - Dealer draws cards until he reaches points exceeding 17 which would return the obtained value. Or when the points exceed 21, resulting in the dealer's loss.

CODE:

```
ENEMY_MOVE PROC NEAR


MOV    CX, 0FH                   ; hit until enemy reaches 17
MOV    DX, 4240H                 ; or until dealers value overpasses the player's
MOV    AH, 86H                   ; but must not overpass 21
INT    15H

E_HIT:
   CALL GIVE_CARD                ; call give card
```

```
   CMP SCOREHOLDER, 21          ; check if over 21
   JG WIN

    CMP SCOREHOLDER, 17          ; check if over 17
    JG __ret                     ; if so save the value
    JMP ENEMY_MOVE               ; else hit again


__ret:
   MOV BL, SCOREHOLDER          ; take the current score and save to BL
   MOV ENEMY_CARDVAL, BL         ; save the score value to the dealers points
  RET
ENEMY_MOVE ENDP
```

- **Randomize**
  - Randomly generates a number from 1 to 52, this corresponds to the cards in a standard deck

CODE:

```
RANDOMIZE PROC NEAR

RANDSTART:
   MOV AH, 00h              ; interrupts to get system time
   INT 1AH                  ; CX:DX now hold number of clock ticks since midnight


        mov  ax, dx
        xor  dx, dx
        mov  cx, 12
        div  cx             ; here dx contains the remainder of the division - from 0 to 14
RET
RANDOMIZE ENDP
```

- **Give Card**
  - Takes the randomly generated number and gives the card that is bound to the random number generated.

CODE:

```
GIVE_CARD PROC NEAR

CALL RANDOMIZE              ; generate a random number
MOV RAND_NUM, DL            ; save randomly generated number

CMP RAND_NUM, 0             ; ace is drawn
JE DISPLAY_ACE

CMP RAND_NUM, 1             ; two is drawn
JE DISPLAY_2
```

```
CMP RAND_NUM, 2          ; three is drawn
JE DISPLAY_3

CMP RAND_NUM, 3          ; four is drawn
JE DISPLAY_4

CMP RAND_NUM, 4          ; five is drawn
JE DISPLAY_5

CMP RAND_NUM, 5          ; six is drawn
JE DISPLAY_6

CMP RAND_NUM, 6          ; seven is drawn
JE DISPLAY_7

CMP RAND_NUM, 7          ; eight is drawn
JE DISPLAY_8

CMP RAND_NUM, 8          ; nine is drawn
JE DISPLAY_9

CMP RAND_NUM, 9          ; ten is drawn
JE DISPLAY_10

CMP RAND_NUM, 10         ; jack is drawn
JE DISPLAy_JACK

CMP RAND_NUM, 11         ; queen is drawn
JE DISPLAY_QUEEN

CMP RAND_NUM, 12         ; king is drawn
JE DISPLAY_KING

RET
GIVE_CARD ENDP
```

- **Displays**
  - Display an individual card from a standard deck from Ace to King, and adds their value to the current points of either the player or the dealer.

CODE:

```
DISPLAY_ACE PROC NEAR
        ADD SCOREHOLDER, 10      ; store the card value
         LEA DX, H_ACE
        CALL DISPLAY_STR          ; display the card
        RET
DISPLAY_ACE ENDP
;-------------------------------------------------------------
DISPLAY_2 PROC NEAR
        ADD SCOREHOLDER, 2       ; store the card value
```

```
        LEA DX, C_TWO
        CALL DISPLAY_STR        ; display the card
        RET
DISPLAY_2 ENDP
;-------------------------------------------------------------
DISPLAY_3 PROC NEAR
        ADD SCOREHOLDER, 3      ; store the card value
        LEA DX, C_THREE
        CALL DISPLAY_STR        ; display the card
         RET
DISPLAY_3 ENDP
;-------------------------------------------------------------
DISPLAY_4 PROC NEAR
        ADD SCOREHOLDER, 4      ; store the card value
        LEA DX, C_FOUR
        CALL DISPLAY_STR        ; display the card
        RET
DISPLAY_4 ENDP
;-------------------------------------------------------------
DISPLAY_5 PROC NEAR
        ADD SCOREHOLDER, 5      ; store the card value
        LEA DX, C_FIVE
        CALL DISPLAY_STR        ; display the card
        RET
DISPLAY_5 ENDP
;-------------------------------------------------------------
DISPLAY_6 PROC NEAR
        ADD SCOREHOLDER, 6      ; store the card value
        LEA DX, C_SIX
        CALL DISPLAY_STR         ; display the card
        RET
DISPLAY_6 ENDP
;-------------------------------------------------------------
DISPLAY_7 PROC NEAR
        ADD SCOREHOLDER, 7      ; store the card value
        LEA DX, C_SEVEN
        CALL DISPLAY_STR        ; display the card
        RET
DISPLAY_7 ENDP
;-------------------------------------------------------------
DISPLAY_8 PROC NEAR
        ADD SCOREHOLDER, 8      ; store the card value
        LEA DX, C_EIGHT
        CALL DISPLAY_STR        ; display the card
        RET
DISPLAY_8 ENDP
;-------------------------------------------------------------
DISPLAY_9 PROC NEAR
         ADD SCOREHOLDER, 9      ; store the card value
        LEA DX, C_NINE
```

```
        CALL DISPLAY_STR        ; display the card
        RET
DISPLAY_9 ENDP
;---------------------------------------------------------------
DISPLAY_10 PROC NEAR
        ADD SCOREHOLDER, 10     ; store the card value
        LEA DX, C_TEN
        CALL DISPLAY_STR        ; display the card
        RET
DISPLAY_10 ENDP
;---------------------------------------------------------------
DISPLAY_JACK PROC NEAR
        ADD SCOREHOLDER, 10     ; store the card value
        LEA DX, C_JACK
        CALL DISPLAY_STR        ; display the card
        RET
DISPLAY_JACK ENDP
;---------------------------------------------------------------
DISPLAY_QUEEN PROC NEAR
        ADD SCOREHOLDER, 10     ; store the card value
        LEA DX, C_QUEEN
        CALL DISPLAY_STR        ; display the card
        RET
DISPLAY_QUEEN ENDP
;---------------------------------------------------------------
DISPLAY_KING PROC NEAR
        ADD SCOREHOLDER, 10     ; store the card value
        LEA DX, C_KING
        CALL DISPLAY_STR        ; display the card
        RET
DISPLAY_KING ENDP
```

- **Clear Screen**
  - Clears the screen with the default colors of dosbox for background and foreground

CODE:

```
_CLEAR_SCREEN PROC NEAR
        MOV   AX, 0600H
        MOV   BH, 07H           ; default dosbox colors
        MOV   CX, 0000H         ; from index (0,0)
        MOV   DX, 184FH         ; to index (24, 79)
        INT   10H
        RET
_CLEAR_SCREEN ENDP
```

- **Display Str**
  - Used for saving one line of code when needing to display using the LEA syntax

CODE:

```
DISPLAY_STR PROC NEAR
        MOV AH, 09H          ; used to shorten code by 1 line per print method
        INT 21H
        RET
DISPLAY_STR ENDP
```

- **Accept Str1**
  - Listener for user inputs during gameplay. Listens for choices of either hit or stand

CODE:

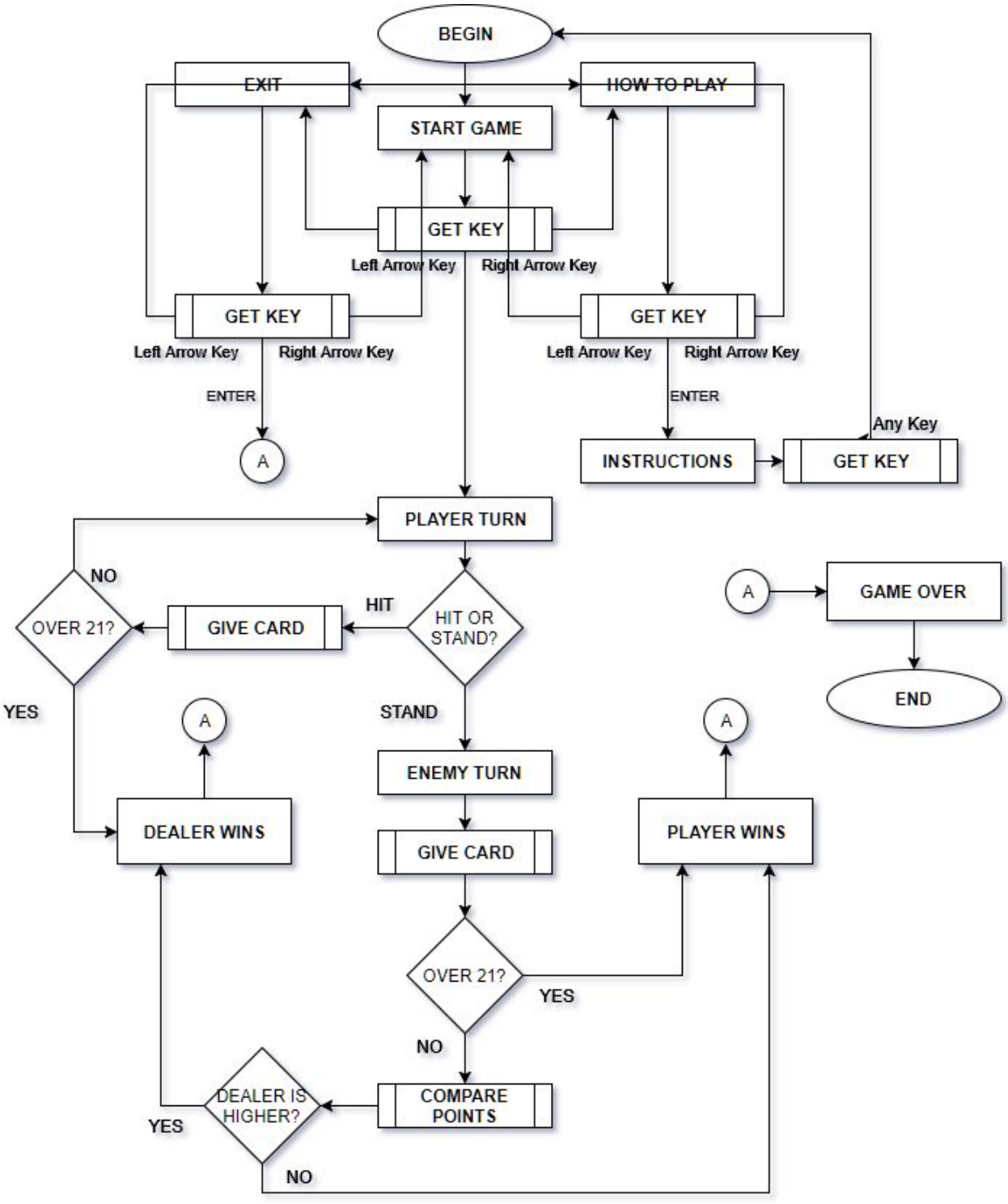
```
ACCEPT_STR1 PROC  NEAR
        MOV AH, 3FH                ; read input of player
        MOV BX, 00
        MOV CX, MAXLEN             ; max length of 30
        LEA DX, KBINPUT1           ; display input in real time
        INT 21H
        RET
ACCEPT_STR1 ENDP
```

- **Terminate**
  - Ends the program and stops the game completely. Leaves a message for the users to read after terminating

CODE:

```
_TERMINATE PROC NEAR
        MOV   AX, 0600H            ; clear screen
        MOV   BH, 02H              ; green foreground with black background
        MOV   CX, 0000H            ; index (0, 0)
        MOV   DX, 184FH            ; to index (24, 79)
        INT   10H

        MOV   DL, 20               ; set cursor so that the message would be at the center
        MOV   DH, 12
        CALL  _SET_CURSOR

        LEA DX, TERMINATE_STR      ; display game over screen is selected
        MOV AH, 09
        INT 21H

        MOV   DL, 0                ; set the cursor to the bottom of the screen
        MOV   DH, 24
        CALL  _SET_CURSOR

        MOV   AX, 4C00H            ; terminate program
        INT   21H
_TERMINATE ENDP
```

Procedure Flowchart

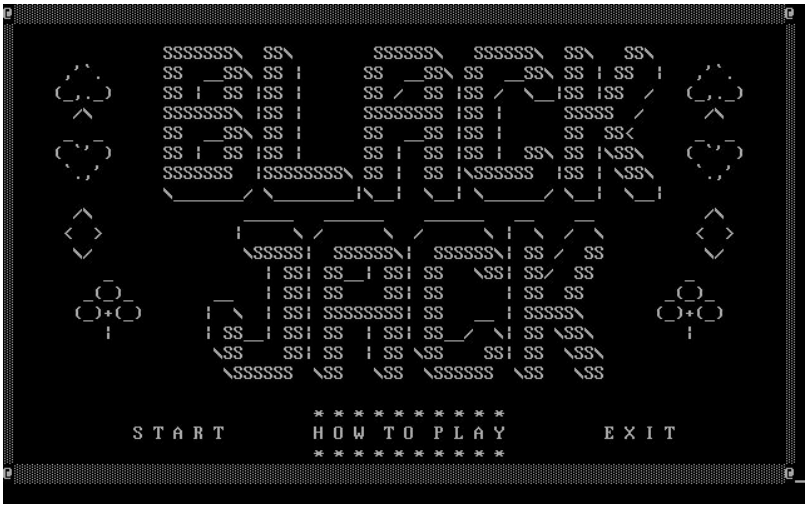


GAME SCREENSHOTS

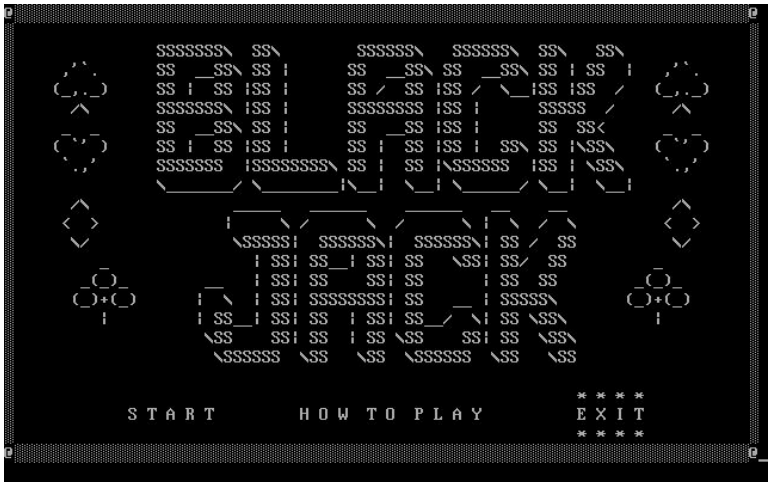
Main Menu

Start is selected display (default display on run)

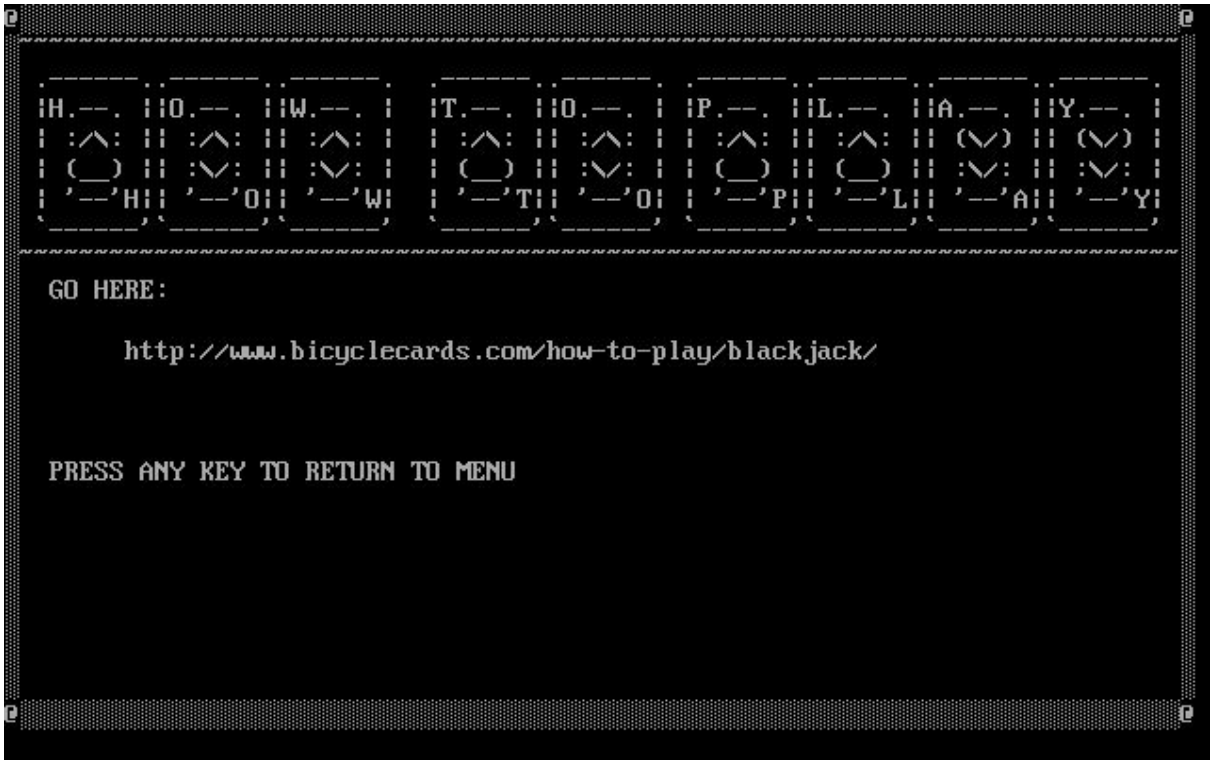
How to play is selected (as of 12/12/17)



Exit is selected (as of 12/12/17)

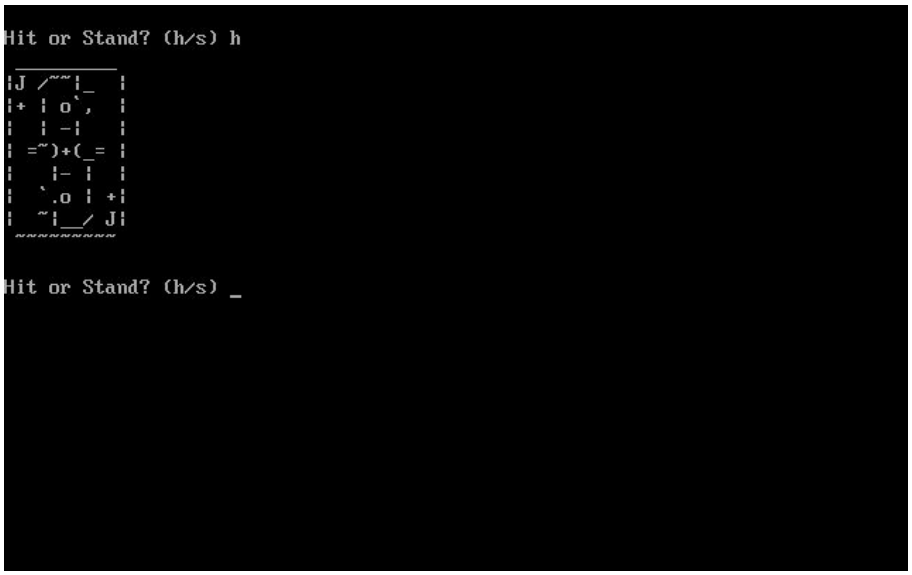


Current how to play display (as of 12/12/17)

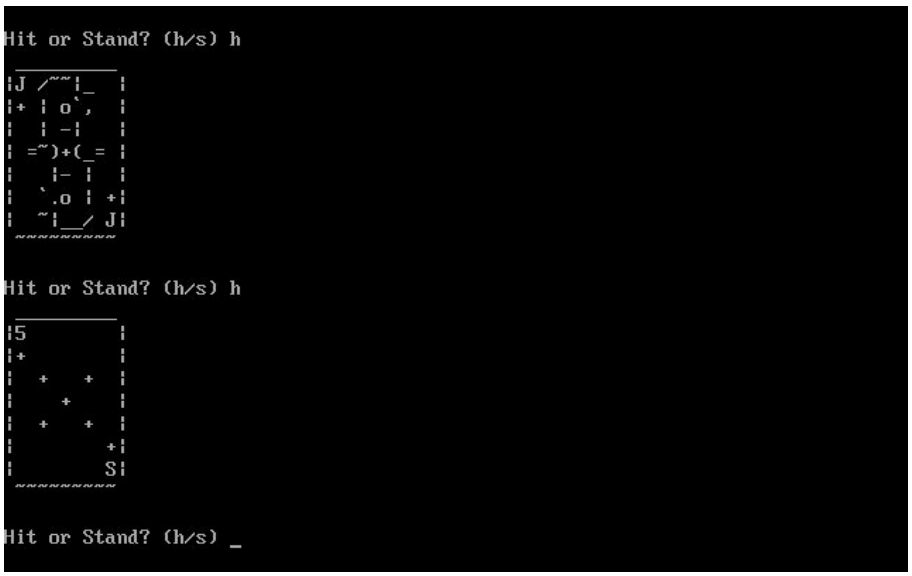


Play State

A choice of either hitting or standing is shown



Player chooses to hit, and a Jack is given. Current player point of 10.



Player hits again, and a 5 is given. Current player points totaling 15.

Player hits again, and receives a 2. Current player points totals to 17.

Player stands, and it is the dealer's turn to play. The dealer draws an Ace. Current dealer points is 10.

The dealer draws another card, and receives a 7. Total points 17. Dealer draws again, and is dealt with an Ace. And has a total of 27 points. Dealer's loss, player wins.

Game Over Screen