



**Área de ingeniería en computadores**  
**Lenguajes, compiladores e intérpretes**

**I tarea programada:**

**BlaCEJack**

**PROFESOR**

Marco Rivera Menénes

**ESTUDIANTES**

Sebastián Mora Godínez 2019227554 [sebastianmg@estudiantec.cr](mailto:sebastianmg@estudiantec.cr)

Emanuel Marín Gutiérrez 2019067500 [emarin702740530@estudiantec.cr](mailto:emarin702740530@estudiantec.cr)

**FECHA**

**19 de marzo, 2021**

## Índice

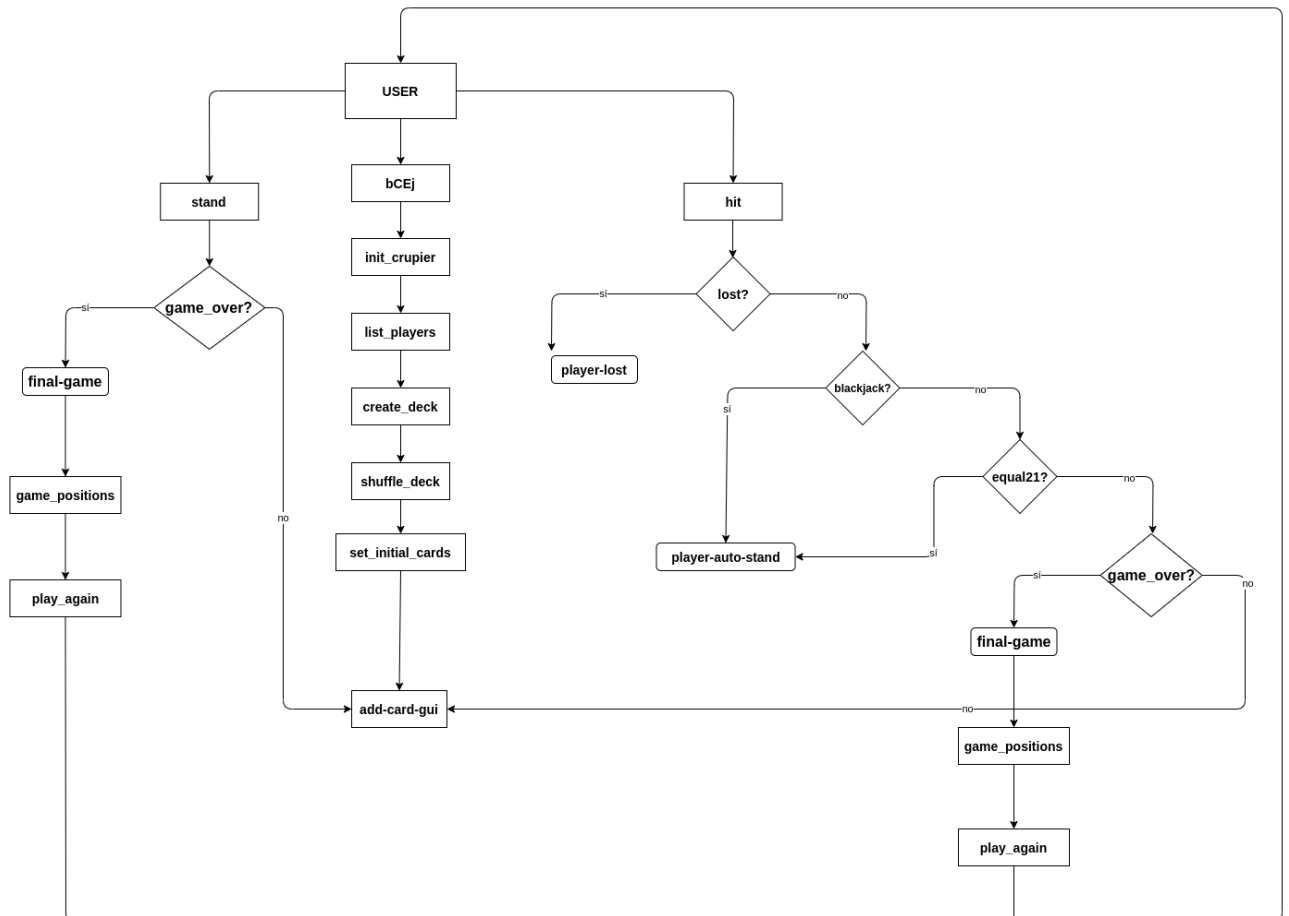
1.1. Descripción de los algoritmos desarrollados.....	3
1.2. Descripción de las funciones implementadas.....	5
1.3. Descripción de las estructuras de datos utilizadas.....	27
1.4. Problemas sin solución.....	29
1.5. Problemas encontrados.....	28
1.6. Plan de actividades.....	30
1.7. Conclusiones.....	33
1.8. Recomendaciones.....	33
1.9. Bibliografía.....	34
1.10. Bitácora de Sebastián Mora.....	35
1.11. Bitácora de Emanuel Marín.....	39

## 1.1 Descripción de los algoritmos de solución desarrollados

El algoritmo general del juego comienza con la función **bCEj** la cual recibe la lista con los nombres de los jugadores para comenzar a inicializar la información necesaria para la partida. Se crea el mazo de cartas, la lista con la información de los jugadores, la lista del crupier y se establece el jugador actual, el cual será el primero de la lista de jugadores. Después se revuelve el mazo de cartas y se reparten dos cartas a cada uno de los jugadores, incluyendo el crupier.

Una vez que comienzan los turnos de los jugadores, estos tienen la opción de pedir una carta o bien, plantearse. En caso de que pidan, se les agrega una nueva carta a su baraja y se verifica primero que la suma de las cartas no haya superado 21, de ser así, su estado se actualiza a *lost* y sigue el siguiente jugador. Por otra parte, si la baraja suma 21 el jugador obtiene un blackjack y automáticamente se plantea para que siga el siguiente jugador. Si no se cumple ninguna de las condiciones anteriores, el jugador puede volver a pedir una carta.

Además, si el jugador decide plantearse, se actualiza su estado a *stand* y se pasa al siguiente jugador. En el caso de que el último jugador pierda o decida plantearse, comienza a jugar el crupier, aplicando las reglas descritas en la especificación. Cada vez que alguno de los jugadores realice alguna de estas acciones, se verifica si el juego terminó para devolver la tabla de posiciones.



Para una mejor visualización del diagrama visite:

<https://github.com/sebas-mora28/BlackJack/blob/master/docs/Diagrama.png>

## 1.2. Descripción de las funciones implementadas

**Nombre:** bCEj\_logic

**Entradas:** **players:** lista con los nombres de los jugadores de la partida

**Salidas:** lista con la información de la partida.

```
(define (bCEj_logic players)
  (set_initial_cards (shuffle_deck (list (create_deck) (list_players players) (init_crupier) 1))))
```

Esta función se encarga de inicializar todos los valores necesarios, agregarles las cartas iniciales a los jugadores y al crupier, revolver el mazo. Devuelve la lista con el mazo, lista de jugadores, crupier, jugador actual.

**Nombre:** stand

**Entradas:** **game\_info:** lista con la información actual de partida.

**id:** id del que realiza la acción stand.

**score:** puntaje del que realiza la acción stan

**Salidas:** La lista del crupier en caso de que sea este el que realiza la acción, de caso contrario devuelve la lista con la información actual de la partida actualizada.

```
(define (stand game_info id score)
  (cond [(= id 0)
        (update_crupier_score_and_state (crupier game_info) "stand" score)]

        [(= id (length (players game_info)))
        (update_game (deck game_info)
                     (update_player_score_and_state (players game_info) id "stand" score)
                     (start_crupier game_info)
                     0)]

        [else
         (update_game (deck game_info)
                      (update_player_score_and_state (players game_info) id "stand" score)
                      (crupier game_info) (+ (current_player_id game_info)
                                              1))]))
```

La función `stand` es la encargada de llevar a cabo la funcionalidad de plantearse.

Se comenzará por su primer condicional. Este verifica si el jugador actual de la partida es igual a 0, es decir, es el crupier el que se encuentra jugando; en caso de ser así entonces llama la función `update_crupier_score_and_state`, la cual devuelve la lista del crupier con el estado y puntaje actualizados.

En la segunda condición, verifica si el id del jugador actual es el mismo que el tamaño de la lista de jugadores, esto es porque el id, corresponde a la posición en la que se encuentra el jugador en la lista de jugadores, por ejemplo, el primer jugador tiene id de 1, el segundo tiene id de 2 y así sucesivamente; por lo tanto, en el momento en que el id corresponda al tamaño de la lista de jugadores, el siguiente en jugar será el crupier, por lo que primeramente se actualizará el estado a `stand` y el puntaje del último jugador llamando la función `update_player_score_and_state`, la cual devuelve la lista de jugadores actualizada, y luego establece como 0 el id del jugador actual, ya que este es el id que corresponde al crupier.

Finalmente, si no sucede ninguna de las condiciones anteriores, ocurre algo parecido a la condición anterior; se actualizarán el estado a `stand` y el puntaje del jugador que realizó la acción de plantearse llamando la función `update_player_score_and_state`, la diferencia radica en que el id del jugador actual se le sumará uno, indicando que ahora existe un nuevo jugador en la partida.

#### Nombre : hit

**Entradas:** `game_info`: lista con la información actual de partida.

`id`: id del jugador que va a realizar la acción hit.

**Salidas:** lista con la información actual de la partida actualizada.

```
(define (hit game_info id)
  (cond [(= id 0)
        (update_game (cdr (deck game_info))
                      (players game_info)
                      (add_card_to_crupier (deck game_info) (crupier game_info))
                      (current_player_id game_info))]
        [else
         (update_game (cdr (deck game_info))
                      (add_card_to_player (deck game_info) (players game_info) id)
                      (crupier game_info)
                      (current_player_id game_info))]))
```

Esta es la función principal que permite tomar una carta a un jugador o al crupier.

En la primera condición se verifica si el id es igual a 0, es decir, si es el crupier el que realiza la acción de tomar una carta, en este caso, se llama la función *update\_game*, la cual actualiza la lista de la información del juego, en el argumento que corresponde al crupier, se llama la función *add\_card\_to\_crupier*, la cual agrega una carta a la baraja del crupier y devuelve la lista de este actualizada.

En caso de que no sea el crupier el que esté realizando la acción, quiere decir que es cualquiera de los jugadores que se encuentran en la partida, en este caso, se llama de igual manera la función *update\_game*, sin embargo, ahora se llamará la función *add\_card\_to\_player* en el argumento donde deben ir la lista de jugadores, esto es porque esta función agrega una carta al jugador indicado por el id y devuelve la lista de jugadores actualizada.

Cabe destacar que las dos ocasiones en que se llama la función *update\_game*, en su primer argumento, donde debe ir el mazo de cartas del juego, se aplica el *cdr* a la lista del mazo, esto es porque al momento de agregar una carta a la baraja ya sea del jugador o del crupier se toma la primera carta de este pues el mazo ya se encuentra desordenado, por lo tanto, para evitar cartas repetidas, se toma la lista del mazo sin el primer elemento.

**Nombre :** *add\_card\_to\_crupier*

**Entradas:** **deck:** baraja del crupier.

**crupier:** lista con la información del crupier.

**Salidas:** lista de la información del crupier actualizada con la nueva carta.

```
(define (add_card_to_crupier deck crupier)
  (list (player_name crupier)
        (add (player_deck crupier)(take_card deck))
        0
        (player_state crupier)
        (player_score crupier)))
```

Esta función se encarga de agregar una carta a la baraja del crupier y devolver la misma lista del crupier actualizada.

Para esto, en la segunda posición de la lista, donde debe ir la baraja, se llama la función *add*, la cual agrega un elemento a una lista, en este caso, la carta, la cual se obtiene de la función *take card*, a la lista que representa la baraja del jugador y deja la demás información igual.

**Nombre:** `add_card_to_player`

**Entradas:** **deck:** baraja del jugador.

**players:** lista de jugadores de la partida.

**id:** id del jugador al que se desea agregar la carta.

**Salidas:** lista con jugadores de la partida actualizada.

```
(define (add_card_to_player deck players id)

  (cond [(null? players)
        '()]

        [(= (player_id (car players)) id)
         (cons (list (player_name(car players))
                     (add (cadar players) (take_card deck))
                     (player_id (car players))
                     (player_state (car players))
                     (player_score (car players))) (add_card_to_player deck (cdr players) id))]

        [else
         (cons (car players) (add_card_to_player deck (cdr players) id))]))
```

Esta función se encarga de agregar una carta a un jugador indicado por el id.

Consiste en recorrer la lista del jugador, en cada llamada recursiva se compara el id del primer elemento de la lista de jugadores con el id indicado por parámetros, cuando lo encuentra, se llama la función *cons*, en su primer argumento, se crea una lista igual para representar a ese jugador, pero en la segunda posición donde debe ir el mazo, se llama la función *add*. Mientras que en el segundo argumento se llama recursivamente la función quitando el primer elemento de la lista de jugadores para continuar.

En caso de que *players* sea nula, se devuelve una lista vacía. Por otra parte, en caso de que el id no coincida, se llama la función *cons*, en su primer argumento, se pone el primer elemento de *players* y en el segundo, se llama la función recursivamente pero sin el primero elemento de la lista *players*; esto para seguir formando la lista jugadores.



**Nombre:** `start_crupier`

**Entradas:** `game_info`: lista con la información actual de partida.

**Salidas:** lista del crupier con la baraja, estado y puntaje actualizado.

```
(define (start_crupier game_info)
  (cond [(lost? (evaluate_deck (player_deck (crupier game_info)) 11))
        (cond [(lost? (evaluate_deck (player_deck (crupier game_info)) 1))
              (update_crupier_score_and_state (crupier game_info) "lost" (evaluate_deck (player_deck (crupier game_info)) 1))]]
        [(equal21? (evaluate_deck (player_deck (crupier game_info)) 1))
         (update_crupier_score_and_state (crupier game_info) "stand" 21)]
        [else (crupier_next_move game_info (evaluate_deck (player_deck (crupier game_info)) 1))]])
  [(equal21? (evaluate_deck (player_deck (crupier game_info)) 11))
   (update_crupier_score_and_state (crupier game_info) "stand" 21)]
  [else (crupier_next_move game_info (evaluate_deck (player_deck (crupier game_info)) 1))]])
```

Esta es la función principal para la funcionalidad del crupier. Se encarga de buscar el mejor valor para el As, verificar si el crupier perdió o sumó 21 puntos, o bien, llamar la función que establece la siguiente jugada del crupier.

Primeramente se verifica si evaluando la baraja del crupier, con el As valiendo como 11, el puntaje se pasa de 21, en caso de ser, realiza la misma verificación pero con el As valiendo como 1. Si se el puntaje se pasa de 21, no importa el valor que tome el As, quiere decir que el crupier perdió la partida, por lo tanto se llama la función *update\_crupier\_score\_and\_state*, la cual establece el estado del crupier como *lost* y le asigna el puntaje, devolviendo la lista del crupier actualizada. En caso de que el puntaje sume igual a 2, se llama la misma función *update\_crupier\_score\_and\_state*, pero se establece como estado *stand* y le asigna el puntaje de 21. Finalmente, si no se cumple ninguna de estas condiciones, entonces se llama la función *crupier\_next\_move*, la cual se explicará más adelante.

En caso de que la primera condición no se cumpliera, es decir, que la baraja sume menos de 21, con el As valiendo como 11, entonces se realiza las mismas verificaciones que se mencionaron anteriormente, primero se verifica si la baraja suma igual a 21, o bien, se llama la función *crupier\_next\_move*, en caso de que no se cumpla ninguna de las condiciones.

**Nombre:** `crupier_next_move`

**Entradas:** `game_info`: lista con la información actual de la partida  
`score`: puntaje actual de la baraja del crupier.

**Salidas:** lista del crupier actualizada.

```
(define (crupier_next_move game_info score)
  (cond [(>= score 17)
        (update_crupier_score_and_state (crupier game_info) "stand" score)]
        [else
         (start_crupier (hit game_info 0))]))
```

Esta función forma parte de la funcionalidad del crupier.

Esta función se encarga de escoger la siguiente jugadas del crupier dependiendo de la situación en que se encuentre. Si el puntaje que tiene el crupier es mayor o igual que 17, entonces se llama la función *stand*. De no ser así, se llama nuevamente la función *start\_crupier*, pero en su argumento se llama se realiza la acción hit indicando que la va a realizar el id 0, para que devuelva la lista del juego actualizada.

**Nombre:** `game_over?`

**Entradas:** `game_info`: lista con la información actual de la partida.

**Salidas:** true en caso de que el juego ya haya terminado, false en caso contrario.

```
(define (game_over? game_info)
  (= (current_player_id game_info) 0))
```

Esta función se encarga de verificar si el juego ya terminó, para esto verifica si el id del jugador actual es 0, es decir, si el crupier es el jugador actual.

**Nombre:** `next_player`

**Entradas:** `game_info`: lista con la información actual de la partida.

`new_state`: nuevo estado del jugador o crupier.

`score`: puntaje del jugador o crupier.

**Salidas:** lista con la información actual de la partida actualizada, excepto si el jugador actual es el crupier, si se da este caso, se devuelve la lista del crupier actualizada

```
(define (next_player game_info new_state score)
  (cond [(= (current_player_id game_info) 0)
        (list (player_name (crupier game_info))
              (player_deck (crupier game_info))
              0
              new_state
              score)]

        [(= (current_player_id game_info) (length (players game_info)))
         (update_game (deck game_info)
                      (update_player_score_and_state(players game_info) (current_player_id
game_info) new_state score)
                      (start_crupier game_info)
                      0)]

        [else
         (update_game (deck game_info)
                      (update_player_score_and_state(players game_info) (current_player_id
game_info) new_state score)
                      (crupier game_info)
                      (+ (current_player_id game_info) 1))]))
```

Esta función se encarga de establecer el siguiente jugador de la partida, en caso de que el jugador actual perdió. Tiene como parámetros: `game_info` (lista con la información de la partida), `new_state` (nuevo estado del jugador actual).

**Nombre :** `update_crupier_state_and_score`

**Entradas:** **crupier:** lista del crupier.  
**new\_state:** nuevo estado del crupier.  
**score:** puntaje del crupier.

**Salidas:** lista del crupier actualizada.

```
(define (update_crupier_score_and_state crupier new_state score)
  (list (player_name crupier) (player_deck crupier) 0 new_state score))
```

Esta función se encarga de actualizar el puntaje y estado del crupier. Tiene como parámetros: *crupier* (lista que representa el crupier), *new\_state* (nuevo estado del crupier), *score* (puntaje del crupier). Crea una nueva lista que represente al crupier, con mismos datos excepto por el estado y puntaje los cuales son cambiados por los nuevos valores obtenidos desde los parámetros.

**Nombre:** `update_player_score_and_state`

**Entradas:** **players:** lista de jugadores.  
**id:** id del jugador al que se desea actualizar los valores.  
**new\_state:** nuevo estado del jugador.  
**score:** puntaje del jugador.

**Salidas:** lista de jugadores actualizada.

```
(define (update_player_score_and_state players id new_state score)
  (cond [(null? players) '()]

        [(= (player_id (car players)) id)
         (cons (list (player_name (car players)) (player_deck (car players)) (player_id
 (car players)) new_state score)
               (update_player_score_and_state (cdr players) id new_state score)) ]

        [else
         (cons (car players) (update_player_score_and_state (cdr players) id new_state
 score))]])
```

Esta función se encarga de actualizar el puntaje y estado de un jugador específico.

Recorre la lista de jugadores, en cada llamada recursiva compara el id del primer elemento de la lista de jugadores con el recibido por parámetros, en caso de ser así, llama la función *cons*, donde en el primer argumento se crea una lista que representa a ese jugador con los mismos datos excepto por el estado y el puntaje; y en el segundo argumento la llamada

recursiva de la función para continuar conformando la nueva lista de jugadores. En el momento en que la lista es nula, devuelve una lista vacía, por otra parte, si los id no coinciden, se llama la función *cons*, con el primer elemento de la lista de jugadores y la llamada recursiva de la función, recortando este primer elemento a la lista de jugadores.

**Nombre:** `evaluate_deck`

**Entradas:** `deck`: baraja de cartas.

`As_value`: valor del As elegido por el jugador.

**Salidas:** un entero indicando el valor de la baraja después de evaluar todas las cartas.

```
(define (evaluate_deck deck As_value)
  (cond [(null? deck) 0]
        [(number? (caar deck)) (+ (caar deck) (evaluate_deck (cdr deck) As_value))]
        [(equal? (caar deck) "J") (+ 10 (evaluate_deck (cdr deck) As_value))]
        [(equal? (caar deck) "Q") (+ 10 (evaluate_deck (cdr deck) As_value))]
        [(equal? (caar deck) "K") (+ 10 (evaluate_deck (cdr deck) As_value))]
        [(equal? (caar deck) "A") (+ As_value (evaluate_deck (cdr deck) As_value))]
        [else (evaluate_deck (cdr deck) As_value)]))
```

Esta función se encarga de obtener los puntos que suma la baraja de un jugador. Recorre la lista que representa la baraja de cartas, sumando los valores que representa cada una de estas. Si la carta tiene un valor numérico, entonces suma este mismo valor. En caso de ser una letra, suma 10. Por último, si la baraja contiene un As, entonces se sumará el valor elegido por el jugador o crupier.

**Nombre:** `init_crupier`

**Entradas:** No tiene entradas.

**Salidas:** lista con la información del crupier.

```
(define (init_crupier) (list "crupier" (list ) 0 "playing" 0))
```

Esta función inicializa la función del crupier. Devuelve la lista que representa al crupier.

**Nombre:** `init_deck`

**Entradas:** No tiene entradas.

**Salidas:** Una lista que contiene sublistas que representan cada una de las 52 del mazo.

```
(define (create_deck)
  (list '(2 "Clubs") '(2 "Diamonds") '(2 "Hearts") '(2 "Spades")
        '(3 "Clubs") '(3 "Diamonds") '(3 "Hearts") '(3 "Spades")
        '(4 "Clubs") '(4 "Diamonds") '(4 "Hearts") '(4 "Spades")
        '(5 "Clubs") '(5 "Diamonds") '(5 "Hearts") '(5 "Spades")
        '(6 "Clubs") '(6 "Diamonds") '(6 "Hearts") '(6 "Spades")
        '(7 "Clubs") '(7 "Diamonds") '(7 "Hearts") '(7 "Spades")
        '(8 "Clubs") '(8 "Diamonds") '(8 "Hearts") '(8 "Spades")
        '(9 "Clubs") '(9 "Diamonds") '(9 "Hearts") '(9 "Spades")
        '(10 "Clubs") '(10 "Diamonds") '(10 "Hearts") '(10 "Spades")
        '("J" "Clubs") '("J" "Diamonds") '("J" "Hearts") '("J" "Spades")
        '("Q" "Clubs") '("Q" "Diamonds") '("Q" "Hearts") '("Q" "Spades")
        '("K" "Clubs") '("K" "Diamonds") '("K" "Hearts") '("K" "Spades")
        '("A" "Clubs") '("A" "Diamonds") '("A" "Hearts") '("A" "Spades")))
```

Esta función se encarga de inicializar el mazo de cartas. Devuelve una lista con sublista, cada sublista representa una carta.

**Nombre:** `set_initial_cards` y `set_initial_cards_aux`

**Entradas:** `game_info`: lista con la información actual de la partida.

`id`: id del jugador

**Salidas:** lista con la información de la partida actualizada.

```
(define (set_initial_cards game_info)
  (set_initial_cards_aux (set_initial_cards_aux game_info (length (players game_info))) (length (players game_info))))

(define (set_initial_cards_aux game_info id)
  (cond [(= id 0)
        (hit game_info 0)]
        [else
         (set_initial_cards_aux (hit game_info id) (- id 1))]))
```

Esta función y su auxiliar establecen las cartas iniciales tanto para los jugadores como para el crupier.

**Nombre:** `list_players` y `list_players_aux`

**Entradas:** `player_names`: lista con los nombres de los jugadores de la partida dados por los usuarios.

**Salidas:** lista con sublistas que representan la información de los jugadores.

```
(define (list_players players_names) (list_players_aux players_names 1))

(define (list_players_aux players_names num_player)
  (cond [(null? players_names) '()]

        [else (cons (list (car players_names) '() num_player "playing" 0) (list_players_aux
                                                                    (cdr players_names) (+ num_player 1)))]))
```

La primera función recibe por parámetros la lista con los nombres de los jugadores, y después llama a su auxiliar.

La función auxiliar recorre la lista de nombres de jugadores para ir formando una lista para cada jugador, al mismo tiempo se le asigna el id correspondiente.

**Nombre:** `shuffle_deck`

**Entradas:** `game_info`: lista con la información actual de la partida.

**Salidas:** lista con la partida actual de la partida actualizada, el mazo de cartas ahora se encuentra revuelto.

```
(define (shuffle_deck game_info) (cons (shuffle (car game_info)) (cdr game_info)))
```

Esta función se encarga de revolver el mazo de cartas. Tiene como parámetros: `game_info` (lista con la información de la partida). Para revolver el mazo se utiliza la función de racket `shuffle`.

**Nombre:** deck

**Entradas:** **game\_info:** lista con la información actual de la partida.

**Salidas:** lista que representa el mazo de cartas de la partida.



```
(define (deck game_info) (car game_info))
```

Esta función devuelve el mazo de cartas de la lista que contiene la información de la partida actual.

**Nombre:** players

**Entradas:** **game\_info:** lista con la información actual de la partida.

**Salidas:** lista que contiene a cada uno de los jugadores de la partida.



```
(define (players game_info) (cadr game_info))
```

Esta función se encarga de devolver la lista de jugador de la lista que contiene la información de la partida actual.

**Nombre:** crupier

**Entradas:** **game\_info:** lista con la información actual de la partida.

**Salidas:** lista que representa el crupier.



```
(define (crupier game_info)(caddr game_info))
```

Esta función se encarga de devolver la lista del crupier de la lista que contiene la información de la partida actual.



**Nombre:** `current_player_id`

**Entradas:** `game_info`: lista con la información actual de la partida.

**Salidas:** entero que representa el id del jugador actual.



```
(define (current_player_id game_info) (caddr game_info))
```

Esta función se encarga de devolver el id del jugador actual de la lista que contiene la información de la partida actual.

**Nombre:** `player_id`

**Entradas:** `* player`: lista que representa a un jugador en específico.

**Salidas:** entero que representa el id del jugador.



```
(define (player_id player) (caddr player))
```

Esta función se encarga de devolver el id del jugador. Tiene como parámetros: *player* (lista que representa a un jugador específico).

**Nombre:** `player_deck`

**Entradas:** `player`: lista que representa a un jugador en específico.

**Salidas:** lista que representa la barajas de cartas de un jugador.



```
(define (player_deck player) (cadr player))
```

Esta función se encarga de devolver el mazo del jugador. Tiene como parámetros: *player* (lista que representa a un jugador específico).

**Nombre:** `player_name`

**Entradas:** `player`: lista que representa a un jugador en específico.

**Salidas:** string que representa el nombre del jugador.

```
(define (player_name player) (car player))
```

Esta función se encarga de devolver el nombre del jugador. Tiene como parámetros: *player* (lista que representa a un jugador específico).

**Nombre:** `player_state`

**Entradas:** `player`: lista que representa a un jugador en específico.

**Salidas:** string que representa el estado del jugador.

```
(define (player_state player) (caddr player))
```

Esta función se encarga de devolver el estado del jugador. Tiene como parámetros: *player* (lista que representa a un jugador específico).

**Nombre:** `player_score`

**Entradas:** `player`: lista que representa a un jugador en específico.

**Salidas:** entero que representa el puntaje del jugador.

```
(define (player_score player) (car (cddddr player)))
```

Esta función se encarga de devolver el puntaje del jugador. Tiene como parámetros: *player* (lista que representa a un jugador específico).

**Nombre:** `take_card`

**Entradas:** `deck`: lista que representa el mazo de cartas.

**Salidas:** lista que representa una carta.

```
(define (take_card deck) (car deck))
```

Esta función se encarga de tomar una carta del mazo de carta, devuelve la primera carta del mazo ya que este se encuentra desordenado.

**Nombre:** `current_player`

**Entradas:** `game_info`: lista con la información actual de la partida.

**Salidas:** lista que representa el jugador actual de la partida.

```
(define (current_player game_info) (player_by_id game_info (current_player_id game_info)))
```

Esta función devuelve la lista del jugador actual. Para eso, primero obtiene el id del jugador actual mediante la función `current_player_id` y luego llama la función `player_by_id` y le ingresa ese id.

**Nombre:** `update_game`

**Entradas:** `deck`: lista que representa el mazo del juego.

`players`: lista que representa los jugadores de la partida.

`crupier`: lista que representa el crupier.

`current_player`: entero que representa el jugador actual de la partida.

**Salidas:** lista con la información de la partida actualizada.

```
(define (update_game deck players crupier current_player) (list deck players crupier current_player))
```

Esta función se encarga de actualizar la lista con la información juego. Tiene como parámetros cada uno de sus partes: *deck*, *players*, *crupier*, *current\_player*.

**Nombre:** `player_id` y `player_id_aux`

**Entradas:** `game_info`: lista con la información actual de la partida.

`id`: id del jugador que se desea buscar.

`players`: lista de los jugadores de la partida.

**Salidas:** lista del jugador que coincide con el id buscado.

```
(define (player_by_id game_info id) (player_by_id_aux (players game_info) id))

(define (player_by_id_aux players id)
  (cond [(null? players) (raise "Player not found")]
        [(= (player_id (car players)) id) (car players)]
        [else (player_by_id_aux (cdr players) id)]))
```

La primera función se encarga de obtener la lista de jugadores de *game\_info* y llamar a su auxiliar.

La función auxiliar recorre la lista de jugadores, en cada llamada compara el id del primer elemento actual de la lista de jugadores con el id dado por parámetros, en caso de ser iguales, devuelve la lista de ese jugador.

**Nombre:** `add`

**Entradas:** `lista`: lista a la cual se desea agregar el elemento.

`elem`: elemento que se desea agregar

**Salidas:** lista actualizada con el nuevo elemento

```
(define (add lista elem)
  (cond [(null? lista) (list elem)]
        [else (cons (car lista) (add (cdr lista) elem))]))
```

Esta función se encarga de agregar un nuevo elemento al final de una lista.

**Nombre:** `orden_by_amout_cards`

**Entradas:** **players:** lista jugadores a ordenar, puede estar incluido el crupier.

**Salidas:** lista con los jugadores ordenados por su cartas.

```
(define (orden_by_amount_cards players)
  (cond [ (null? players) '()]
        [else
         (append (orden_by_amount_cards (less_than_cards (car players) (cdr players)))
                   (list (car players))
                   (orden_by_amount_cards (greater_than_cards (car players) (cdr players))))]))
```

Esta función se encarga de ordenar una lista de jugadores por su cantidad de cartas. Utiliza el método de ordenamiento de quicksort. Tiene como parámetros: *players* (lista de jugadores, puede incluir el crupier)

**Nombre:** `orden_by_score`

**Entradas:** **players:** lista jugadores a ordenar, puede estar incluido el crupier.

**Salidas:** lista con los jugadores ordenados por su puntaje.

```
(define (orden_by_score players)
  (cond [ (null? players) '()]
        [else
         (append (orden_by_score (less_than_score (car players) (cdr players)))
                   (list (car players))
                   (orden_by_score (greater_than_score (car players) (cdr players))))]))
```

Esta función se encarga de ordenar una lista de jugadores por su puntaje. Utiliza el método de ordenamiento de quicksort. Tiene como parámetros: *players* (lista de jugadores, puede incluir el crupier).

**Nombre:** `greater_then_cards`

**Entradas:** **players:** lista jugadores, puede estar incluido el crupier.

**Salidas:** lista con los jugadores que tienen una mayor cantidad de cartas que el pivote.

```
(define (greater_than_cards pivot players)
  (cond [(null? players) '()]

        [(< (length (player_deck (car players))) (length (player_deck pivot))) (greater_than_cards pivot (cdr players))]
        [else
         (cons (car players) (greater_than_cards pivot (cdr players)))]))
```

Esta función se encarga de buscar los jugadores que tienen una cantidad de cartas mayor que el pivote. Recorre la lista de jugadores comparando el tamaño de la lista que representa la baraja de cartas. Tiene como parámetros: *pivot* (pivote), *players* (lista de jugadores, puede incluir el crupier).

### Nombre: **greater\_then\_score**

**Entradas:** **players:** lista jugadores, puede estar incluido el crupier.

**Salidas:** lista con los jugadores que tienen un mayor puntaje que el pivote.

```
(define (greater_than_score pivot players)
  (cond [(null? players) '()]

        [(< (player_score (car players)) (player_score pivot)) (greater_than_score pivot (cdr players))]
        [else
         (cons (car players) (greater_than_score pivot (cdr players)))]))
```

Esta función se encarga de buscar los jugadores que tienen un puntaje mayor que el pivote. Recorre la lista de jugadores comparando el tamaño de la lista que representa la baraja de cartas. Tiene como parámetros: *pivot* (pivote), *players* (lista de jugadores, puede incluir el crupier).

### Nombre: **less\_then\_cards**

**Entradas:** **players:** lista jugadores, puede estar incluido el crupier.

**Salidas:** lista con los jugadores que tienen una cantidad de cartas menor que el pivote.

```
(define (less_than_cards pivot players)
  (cond [(null? players) '()]

        [(>= (length (player_deck (car lista))) (length (player_deck pivot))) (less_than_cards pivot (cdr players))]
        [else
         (cons (car players) (less_than_cards pivot (cdr players)))]))
```

Esta función se encarga de buscar los jugadores que tienen una cantidad de cartas menor que el pivote. Recorre la lista de jugadores comparando el tamaño de la lista que representa la baraja de cartas. Tiene como parámetros: *pivot* (pivote), *players* (lista de jugadores, puede incluir el crupier).

**Nombre:** `less_than_score`

**Entradas:** **players:** lista jugadores, puede estar incluido el crupier.

**Salidas:** lista con los jugadores que tienen un puntaje menor que el pivote.

```
(define (less_than_score pivot players)
  (cond [(null? players) '()]

        [(>= (player_score (car players)) (player_score pivot)) (less_than_score pivot (cdr players))]

        [else
         (cons (car players) (less_than_score pivot (cdr players)))]))
```

Esta función se encarga de buscar los jugadores que tienen un puntaje menor que el pivote. Recorre la lista de jugadores comparando el tamaño de la lista que representa la baraja de cartas.

**Nombre:** `losers`

**Entradas:** **players:** lista de perdedores.

**Salidas:** lista con los jugadores perdieron.

```
(define (losers players)
  (cond [(null? players) '()]

        [(equal? (player_state (car players)) "lost")
         (cons (car players) (losers (cdr players)))]

        [else (losers (cdr players))]))
```

Esta función se encarga de filtrar aquellos jugadores que tienen como estado *lost*. Recorre la lista verificando el estado de cada jugador.

**Nombre:** `players_in_game`

**Entradas:** `players`: lista de ganadores.

**Salidas:** lista con los jugadores que todavía no hayan perdido.

```
(define (players_in_game players)
  (cond [(null? players) '()]

        [(equal? (player_state (car players)) "stand")
         (cons (car players) (players_in_game (cdr players)))]

        [else (players_in_game (cdr players))]))
```

Esta función se encarga de filtrar aquellos jugadores que tienen como estado *stand*. Recorre la lista verificando el estado de cada jugador.

**Nombre:** `ord_possible_winner` y `ord_losers`

**Entradas:** `losers`: lista con los jugadores que perdieron durante la partida.

`possibles_winners`: lista con los posibles ganadores de la partida.

**Salidas:** lista actualizada con el nuevo elemento.

```
(define (ord_losers losers)
  (orden_by_score (orden_by_amount_cards losers)))

(define (ord_possible_winners posibles_winners)
  (reverse (orden_by_score (reverse (orden_by_amount_cards posibles_winners)))))
```

La primera función ordena la lista de jugadores que perdieron, primero por su cantidad de cartas, y luego por su puntaje.

La segunda función ordena la lista de posibles ganadores, primero por su cantidad de cartas, y luego por su puntaje. A la lista se le aplica el reverse para que quede ordenada correctamente.



**Nombre:** `game_positions`

**Entradas:** `game_info`: lista con la información de la partida.

**Salidas:** lista con la tabla de posiciones.

```
(define (game_positions game_info)
  (cond [(equal? (player_state (crupier game_info)) "stand")
        (append (ord_possible_winners (append (players_in_game (players game_info)) (list (crupier game_info))))
                (ord_losers (losers (players game_info))))]
        [else
         (append (ord_possible_winners (players_in_game (players game_info))) (ord_losers (losers (append
           (players game_info) (list (crupier game_info)))))))]])
```

Esta función se encarga de devolver una lista con la tabla de posiciones de la partida actual.

**Nombre:** `lost?`

**Entradas:** `player_score`: puntaje del jugador.

**Salidas:** `true` en caso de que el puntaje sume más de 21, `false` en caso contrario.

```
(define (lost? player_score)
  (> player_score 21))
```

Esta función recibe el puntaje del jugador y verifica si es mayor a 21, es decir, si el jugador perdió. Devuelve `true` en caso de que el puntaje supera 21, `false` en caso contrario.

**Nombre:** `equal21?`

**Entradas:** `player_score`: puntaje del jugador

**Salidas:** `true` en caso de que el puntaje sea igual a 21, `false` en caso contrario.

```
(define (equal21? player_score)
  (= player_score 21))
```

Esta función recibe el puntaje del jugador y verifica si es igual a 21. Devuelve `true` en caso de que el puntaje sea igual a 21, `false` en caso contrario.

**Nombre:** blackjack?

**Entradas:** **player\_score:** puntaje del jugador.

**deck:** baraja del jugador.

**Salidas:** true en caso de que el puntaje sea igual a 21 y tenga dos cartas, false en caso contrario.

```
(define (blackjack? player_score deck)
  (and (= player_score 21) (= (length deck) 2)))
```

Esta función recibe el puntaje del jugador y verifica si es igual a 21 y además que en la baraja solamente tenga dos cartas. Devuelve *true* en caso de que el puntaje sea igual a 21 y tenga dos cartas en la baraja, *false* en caso contrario.

### 1.3. Descripción de las estructuras de datos

En este proyecto se utilizaron principalmente listas de Racket, como veremos a continuación.

- **Crupier**

**Tipo:** lista de racket

**Descripción:** Esta lista representa al crupier en el juego, esta contiene información como el nombre, baraja de cartas, la cual al mismo tiempo se representa como una lista con sublistas, donde cada sublista en una carta individual; estado (playing, stand, lost), id (el cual debe ser 0 por ser el crupier) y puntaje final.



```
(nombre baraja id estado puntaje)
```

```
(crupier ((9 Spades) (A Spades)) 0 "playing" 20)
```

- **Player**

**Tipo:** lista de racket

**Descripción:** Esta lista representa a los jugadores en el juego, esta contiene información como el nombre, baraja de cartas, la cual al mismo tiempo se representa como una lista con sublistas, donde cada sublista en una carta individual; estado, id y puntaje final.




```
((nombre baraja id estado puntaje) (nombre baraja id estado puntaje) ...)
```

```
((Pedro ((4 Hearts) (5 Hearts) (8 Hearts)) 1 stand 17) ( Carlos ((5 Diamonds) (5 Clubs)) 2 stand 10))
```

- **Deck**

**Tipo:** lista de racket

**Descripción:** Esta lista representa el mazo de carta. Está conformada por sublistas, donde cada sublista representa una carta individual.



```
((2 "Diamonds") (6 "Hearts") (7 "Spades") (Q "Clubs") (J "Diamonds") (9 "Hearts") ...)
```

- **Lista con la información del juego, conocida como game\_info**

**Tipo:** lista de Racket

**Descripción:** Lista que representa la información de la partida, contiene el mazo de cartas, la lista de jugadores, el crupier y el jugador actual de la partida.



```
((2 "Clubs") (4 "Hearts")...) ((("Maria" (("A" "Spades") ("K" "Diamonds")) 1 "playing" 0) ("Juan" ...)) ("crupier" ... ) 1)
```

## 1.4. Problemas sin solución

Todos los problemas que se encontraron durante el proyecto fueron solucionados.

## 1.5. Problemas encontrados

- **Importación de archivos:**

Hubo problemas al enlazar la lógica del juego (bCEj.rkt) con la interfaz gráfica (bCEj\_gui.rkt), a pesar que ambos archivos estaban contenidos en el mismo lugar, se consultó en internet y el problema se resolvió al quitarle a la lógica del juego, la línea donde se indica que el archivo usará la variante normal de Racket (`#lang racket`). Se recomienda que el archivo que se desea importar esté en el mismo lugar de aquel que lo va a utilizar y que ambos archivos sean independientes entre sí.

- **Adjuntar imágenes:**

En la interfaz gráfica era indispensable cargar las imágenes del mazo de cartas y los fondos de cada una de las ventanas, no obstante, hubieron ciertos inconvenientes con este apartado, pues no se tenía un medio claro para hacerlo. Consultando la documentación de racket entorno al desarrollo de interfaces gráficas, se encontró que la clase `message%` permite en uno de sus parámetros (`label`), pasar la ruta de la imagen que se desea cargar por medio de un mapa de bits (`read-bitmap`).

Si lo que se busca es adjuntar imágenes estáticas, se recomienda usar este método, por lo contrario, si se necesita manipular variables de la imagen (dimensión, posicionamiento, filtros, entre otros) es imperioso buscar un método alternativo.

- **Eliminar elementos de un contenedor:**

En la interfaz, cuando un jugador perdía o se plantaba era necesario quitar de la interfaz toda la información de este (nombre y cartas) y colocar la información del siguiente jugador. Al principio tuve problemas para eliminar los hijos de un contenedor padre, pero al estudiar la documentación oficial de Racket, pude entender los métodos que estos presentan para lograr tales objetivos. Se recomienda estudiar todas las clases y métodos de ventanas que ofrece el kit de herramientas para interfaces de Racket, pues facilitan la distribución y manejo de los elementos dentro de ellos, permitiendo construir interfaces más amigables con el usuario y de buen diseño estético.

## 1.6. Plan de actividades

Descripción de la tarea	Tiempo estimado	Responsable	Fecha de entrega
Implementar una función que genere el mazo de cartas	1 hora	Sebastián Mora Godínez	Lunes 8 de marzo
Leer la documentación sobre el desarrollo de interfaces gráficas en Racket	2 Horas	Emanuel Marín Gutiérrez	Lunes 8 de marzo
Implementar una función que genere una lista con los jugadores y sus cartas	2 horas	Sebastián Mora Godínez	Martes 9 de marzo
Creación de ventanas, botones, contenedores y sus derivados	1 Hora	Emanuel Marín Gutiérrez	Martes 9 de marzo
Manejo de eventos (mouse y teclado)	30 mins	Emanuel Marín Gutiérrez	Martes 9 de marzo
Implementar función que reparte las cartas iniciales al jugador y crupier.	3 horas	Sebastián Mora Godínez	Martes 9 de marzo
Implementar una función que revuelve el mazo de cartas	30 minutos	Sebastián Mora Godínez	Martes 9 de marzo
Implementar la funcionalidad de tomar una carta	1 día	Sebastián Mora Godínez	Miércoles 10 de marzo
Implementar una función que evalúe el mazo de cartas	1 hora	Sebastián Mora Godínez	Miércoles 10 de marzo
Manejo de archivos, inserción de imágenes y alineación de los mismos	2 Horas	Emanuel Marín Gutiérrez	Miércoles 10 de marzo
Implementar la funcionalidad de plantearse	1 día	Sebastián Mora Godínez	Jueves 11 de marzo
Establecer e implementar las reglas del juego BlackJack	1 día	Sebastián Mora Godínez	Jueves 11 de marzo
Menú de inicio y ventana de juego	1 Hora	Emanuel Marín Gutiérrez	Jueves 11 de marzo
Implementar una función que verifique el número de jugadores ingresados	30 mins	Emanuel Marín Gutiérrez	Jueves 11 de marzo
Creación y distribución de los componentes de la ventana de inicio así como el acceso a las demás ventanas	30 mins	Emanuel Marín Gutiérrez	Jueves 11 de marzo

(ventana de información y del juego)			
Creación y distribución de la ventana de información acerca del desarrollo del juego	30 mins	Emanuel Marín Gutiérrez	Jueves 11 de marzo
Implementar la funcionalidad del crupier	1 día	Sebastián Mora Godínez	Jueves 11 de marzo
Creación y distribución de la ventana del juego	1 Hora	Emanuel Marín Gutiérrez	Viernes 12 de marzo
Implementar funciones que mantengan actualizados las características del juego (el mazo, el crupier, los jugadores, el valor del as, entre otros)	30 min	Emanuel Marín Gutiérrez	Viernes 12 de marzo
Implementar una función que agregue las dos cartas del crupier con la primera de esta boca abajo y las dos primeras cartas del primer jugador	1 Hora	Emanuel Marín Gutiérrez	Sábado 13 de marzo
Implementar una función que agregue una carta cada vez que el jugador decide tomar una nueva (hit)	30 mins	Emanuel Marín Gutiérrez	Sábado 13 de marzo
Implementar una función que cambie al jugador actual, por el siguiente jugador o por el crupier, cada vez que este pierda o se planta (stand)	1 Hora	Emanuel Marín Gutiérrez	Sábado 13 de marzo
Implementar función para cambiar de jugador de actual, verificar si el juego terminó, si se obtuvo blackjack o si el jugador perdió.	3 horas	Sebastián Mora Godínez	Sábado 13 de marzo
Implementar una función para cuando el jugador decide plantarse (stand)	30 mins	Emanuel Marín Gutiérrez	Domingo 14 de marzo
Implementar una función para cuando el jugador debe plantarse automáticamente, esto es cuando la suma de las cartas ya es igual a 21	30 mins	Emanuel Marín Gutiérrez	Domingo 14 de marzo
Implementar una función que agregue sonido a las acciones del jugador o del juego en general	30 mins	Emanuel Marín Gutiérrez	Domingo 14 de marzo
Desarrollar la lógica para el final del juego donde se muestra una tabla de posiciones con los jugadores y su puntuación	1 Hora	Emanuel Marín Gutiérrez	Martes 16 de marzo

Implementar una función que genera la tabla de posiciones de la partida	1 día	Sebastián Mora Godínez	Martes 16 de marzo
Optimizar el código general de la interfaz	2 Horas	Emanuel Marín Gutiérrez	Miércoles 17 de marzo
Mejorar el aspecto visual de la interfaz y la interrelación con el usuario	6 Horas	Emanuel Marín Gutiérrez	Jueves 18 de marzo



## **1.7. Conclusiones**

- Se reafirmó y amplió el conocimiento del paradigma de programación funcional creando una aplicación en el lenguaje de programación Racket.
- Se utiliza el lenguaje de programación Racket para crear una aplicación completa, desde su lógica hasta su interfaz.
- Se utilizaron listas como estructura de datos principal de la aplicación, lo que facilitó el manejo y flujo de la información necesaria para la ejecución del juego.
- Se aplicaron conceptos sobre la programación funcional adquiridos en la clase, lo que facilitó el aprendizaje y entendimiento de este paradigma.

## **1.8. Recomendaciones**

- Planificar y estructurar el proyecto antes de comenzar a programar, el paradigma funcional puede llegar a complicarse si no se cuenta con una buena planificación.
- Evitar hacer aplicaciones con interfaz gráfica usando Racket, ya que su biblioteca gráfica es muy limitada.

## 1.9. Bibliografía

**2 Racket Essentials.** (s. f.). Racket Documentation. Recuperado de <https://docs.racket-lang.org/guide/to-scheme.html>

**The Racket Graphical Interface Toolkit.** (s. f.). Racket Documentation. Recuperado de <https://docs.racket-lang.org/gui/>

**Including an external file in racket.** (2011, 26 enero). Stack Overflow. <https://stackoverflow.com/questions/4809433/including-an-external-file-in-racket/4937138>

## 1.10. Bitácora de Sebastián Mora

**Fecha:** 7 de marzo, **hora:** 3:00 pm

**Participantes:** Sebastián Mora

**Descripción:**

El día de hoy se comienza a idear la funcionalidad del juego, se estructuran posibles funciones necesarias para la implementación de la lógica. Se debe crear una función que inicializa la lista que va a representar el crupier. Además, implementar una función que inicialice el mazo de cartas y la lista de jugadores, revuelva el mazo de cartas y otra que evalúe y obtenga la cantidad de puntos en una mano de cartas dada.

**Fecha:** 8 de marzo, **hora:** 8:00 pm

**Participantes:** Sebastián Mora

**Descripción:**

El día de hoy se comienza a realizar la implementación de las funciones establecidas el día anterior. Primeramente se implementa las funciones que devuelven el mazo con las 52 cartas definidas, la lista con la información (nombre, cartas, id, estado en la partida) de los jugadores y por último la lista con el crupier con su información ( nombre, cartas, id, estado en la partida). Por otra parte, se implementa la función que evalúa una mano de cartas dada, para esto cuenta la cantidad de puntos de cada carta y lo va sumando; para los “As” cuenta cuántos existen en la mano y lo multiplica por 1 o 11 dependiendo del valor dado.

**Fecha:** 10 de marzo, **hora:** 11:00 am

**Participantes:** Sebastián Mora

**Descripción:**

El día de hoy comencé a desarrollar la funcionalidad de pedir una carta. Se encarga de sacar una carta del mazo y agregarla a la baraja del jugador que pidió. Esta parte está compuesta de 4 funciones: hit, add\_card\_to\_player, add\_card\_to\_crupier, add. Más información en la parte **descripción de las funciones implementadas**. Además implementé varias funciones que retornaban detalles como nombre, baraja, id, estado de un jugador específico o del crupier; además de la lista de jugadores, jugador actual, buscar un jugador por su id.

**Fecha:** 11 de marzo, **hora:** 5:00 pm

**Participantes:** Sebastián Mora

**Descripción:**

El día de hoy realicé pequeños cambios en algunos de nombre de funciones. Arreglé varios bugs que me encontré durante la ejecución de pruebas, más información en **problemas encontrados**. Además implementé la funcionalidad de plantearse, esta acción es la que permite a un jugador mantenerse con las cartas que posee en ese momento y que siga el siguiente jugador. Por otra parte, como cambio para el crupier y los jugadores, se cambiará la forma en que se implementó el estado; a este momento se tenía un flag que indicaba si un jugador estaba jugando o no, pero analizándolo se decidió cambiarlo a estados más específicos, ahora tanto el crupier como el jugador contarán con tres estados: playing, stand, lost; esto para diferenciar de manera más fácil que jugadores ya no se encuentran jugando.

**Fecha:** 12 de marzo, **hora:** 7:00 pm

**Participantes:** Sebastián Mora

**Descripción:**

El día de hoy comencé a desarrollar la lógica del crupier siguiendo las indicaciones de la especificación; primeramente se evalúa el valor que debe tener el As en la baraja y después se verifica si el valor de la baraja es más de 17, en caso de ser así, el jugador procede a plantearse y si es menor a 16, entonces toma una carta. Esta función se llamará después de que el último jugador pierda o se plante. Luego, implementé la función `game_over`, la cual determina si el juego terminó, esta recorre la lista de jugadores para verificar si existe alguno que todavía se encuentra en juego.

Realicé otro cambio en el crupier y en los jugadores, ahora la lista de cada uno tendrá almacenado el puntaje final de su baraja de cartas.

La función `game_over` ahora únicamente verifica si el jugador actual es el crupier.

**Fecha:** 13 de marzo, **hora:** 5:00 pm

**Participantes:** Sebastián Mora

**Descripción:**

El día de hoy, comencé a realizar pruebas para la funcionalidad del crupier, al no obtener los resultados esperados procedí a revisar la función.

Realicé algunos cambios en la función start crupier, ahora primeramente se verificará el valor que debe tomar el As, después si la baraja del jugador suma mayor que 21 o 21. En caso de que ninguna de estas dos condiciones suceda, se procede a llamar una función que determina si el crupier debe plantearse o tomar una carta.

Se implementó la función winners, la cual evalúa y determina el ganador de la partida.

El día de hoy prácticamente se terminó de implementar la lógica, a falta de algunos detalles y arreglo de bugs que puedan ocurrir durante la ejecución del programa.

**Fecha:** 14 de marzo, **hora:** 3:00 pm

**Participantes:** Sebastián Mora

**Descripción:**

El día de hoy me dediqué a arreglar diferentes bugs de sintaxis que Emanuel me comunicó que ocurrían durante la ejecución de la interfaz. La mayoría de bugs se concentraron en la función winner. Todos los bugs fueron arreglados. Se comenzó a realizar la parte de la documentación: descripción de las funciones implementadas.

**Fecha:** 15 de marzo, **hora:** 5:00 pm

**Participantes:** Sebastián Mora

**Descripción:**

El día de hoy continué realizando parte de la documentación. Además de eso, se arreglaron algunos bugs que Emanuel me comunicó que estaban ocurriendo, estos errores eran principalmente errores de sintaxis y semántica por lo que fueron resueltos rápidamente.

**Fecha:** 16 de marzo, **hora:** 6:00 pm

**Participantes:** Sebastián Mora

**Descripción:**

El día de hoy se cambió la función de winners, debido a que en la especificación se espera una tabla de posiciones y esta función solamente retorna el ganador. Ahora se implementa la función *game\_positions* que devuelve la tabla de posiciones de la partida. Se sigue desarrollando la documentación.

**Fecha:** 18 de marzo, **hora:** 2:00 pm

**Participantes:** Sebastián Mora

**Descripción:**

El día de hoy se terminaron los últimos detalles de la tarea, se solucionaron pequeños bugs de sintaxis y semántica. Se terminó la documentación técnica. Además se realizaron pruebas para probar el juego.

## 1.11. Bitácora de Emanuel Marín

**Fecha:** 7 de marzo, **hora:** 5:00 pm

**Participantes:** Emanuel Marín

**Descripción:**

Realicé la lectura de la documentación oficial de Racket para el desarrollo de interfaces gráficas, hice énfasis en las secciones necesarias para la implementación de la tarea (ventanas, widgets, manejo de evento y de archivos, entre otros). La información la complementé con ejemplos y consultas en otras páginas web referentes a Racket.

**Fecha:** 8 de marzo, **hora:** 11:00 am

**Participantes:** Emanuel Marín

**Descripción:**

El día de hoy puse en práctica, mediante pequeños ejemplos, lo aprendido sobre el desarrollo de interfaces gráficas en Racket. Se analizan, manipulan y ejecutan códigos específicos, para familiarizarse con las bibliotecas que provee el lenguaje, entender la instanciación y creación de clases con sus respectivos métodos, el manejo de excepciones, control de flujo y eventos, etc.

**Fecha:** 9 de marzo, **hora:** 9:00 pm

**Participantes:** Emanuel Marín

**Descripción:**

El día de hoy, creé el esqueleto de una interfaz funcional a manera de ejemplo para corroborar lo aprendido hasta el momento (creación y distribución de contenedores, manejo de eventos, excepciones, creación e instanciación de clases, entre otros).

**Fecha:** 10 de marzo, **hora:** 8:00 am

**Participantes:** Emanuel Marín

**Descripción:**

Para hoy, se trabaja todavía sobre la interfaz de ejemplo creada el día anterior y se practica sobre ella la inserción de medios audiovisuales. Se procede a crear la base de la interfaz del juego una vez que ya han sido interiorizados los fundamentos del desarrollo de interfaces gráficas en el lenguaje de programación Racket.

**Fecha:** 11 de marzo, **hora:** 3:00 pm

**Participantes:** Emanuel Marín

**Descripción:**

En el transcurso de la tarde trabajé en la ventana de inicio y en el acceso de esta a la ventana de información y del juego, creé un medio para mostrar a los jugadores que empezarán la partida, cuando estos son pasados como argumento en forma de lista junto al nombre de la función principal en la ventana de comandos. Adicionalmente, trabajé en un medio para que una vez empezado el juego se puedan editar el nombre y la cantidad de jugadores corroborando siempre que cumpla con los requisitos de la especificación de la tarea (se debe ingresar al menos el nombre de un jugador y la cantidad de estos no puede ser mayor a tres). También se agregué los sprites de las cartas que la interfaz utilizará.

**Fecha:** 12 de marzo, **hora:** 10:00 am

**Participantes:** Emanuel Marín

**Descripción:**

El día de hoy trabajé principalmente en la ventana del juego y la distribución de los elementos en la misma, decidí que las reglas del juego estarán en la parte izquierda de la ventana para que el jugador sepa en todo momento las opciones disponibles para ganar la partida en contra del crupier.

Por aparte, creé funciones que permitan mantener actualizados todo la información del juego en cuestión; esto es el mazo de cartas, las cartas de los jugadores y del crupier, el estado actual de cada jugador (si se mantiene jugando, si ya perdió o si decidió plantarse) y el jugador actual (jugador que puede tomar una carta o plantarse mientras los demás esperan que este decida qué hacer).

**Fecha:** 13 de marzo, **hora:** 6:00 am

**Participantes:** Emanuel Marín

**Descripción:**

Hoy creé una función que permite, al momento de empezar el juego, visualizar las dos primeras cartas por default del crupier y del primer jugador (si hay más de uno) con la primera carta del crupier boca abajo.

También hice una función que agrega una carta a la interfaz cada vez que el jugador decida tomar una (hit).

Por último, creé funciones que permitan controlar el flujo del juego, es decir cambiar de jugador una vez que el anterior haya perdido o se haya plantado.



**Fecha:** 14 de marzo, **hora:** 11:00 am

**Participantes:** Emanuel Marín

**Descripción:**

El día de hoy ventanas emergentes fueron creadas para indicarle al jugador si ya perdió, si está seguro que quiere plantarse o si debe plantarse (si la suma de cartas es igual a 21). Además agregué sonido a las acciones del jugador (cuando toma una carta o cuando decide plantarse) y del juego en general (repartición de cartas).

**Fecha:** 15 de marzo, **hora:** 8:00 pm

**Participantes:** Emanuel Marín

**Descripción:**

Hoy empecé a trabajar en la conclusión del juego (cuando el último jugador pierde o se planta y empieza a jugar el crupier) para ello, la carta que el crupier tenía boca abajo se muestra y también las cartas que él va tomando, por último se muestra una ventana emergente con una tabla de las posiciones de los jugadores y su puntuación, y la oportunidad de volver a jugar o salirse del juego.

**Fecha:** 17 de marzo, **hora:** 9:00 am

**Participantes:** Emanuel Marín

**Descripción:**

Hoy decidí optimizar el código general de la interfaz; ver qué funciones son innecesarias, cuáles cumplen la misma función, la dependencia entre funciones, la distribución y conexión de los elementos en las ventanas, entre otros. Adicionalmente fui documentando las funciones principales y los elementos que estas contienen para que el entendimiento de las mismas sea más sencillo a la hora de depurar el código en busca de fallos.

**Fecha:** 18 de marzo, **hora:** 3:00 pm

**Participantes:** Emanuel Marín

**Descripción:**

Finalmente, comencé a mejorar los aspectos audiovisuales de la interfaz de manera que tenga un gran atractivo y sea amigable con el usuario, hice pruebas para comprobar que la interfaz funciona con cada una de las acciones que el usuario pueda realizar a lo hora de ejecutar el programa.