Instituto Tecnológico de Costa Rica

Semestre I

2021

BlackCEJack: Documentación técnica

Lenguajes, compiladores e intérpretes

Tarea#1: Programación funcional

Prof. Marco Rivera Meneses

José Alejandro Chavarría Madriz
Natalia González Bermúdez

2019067306

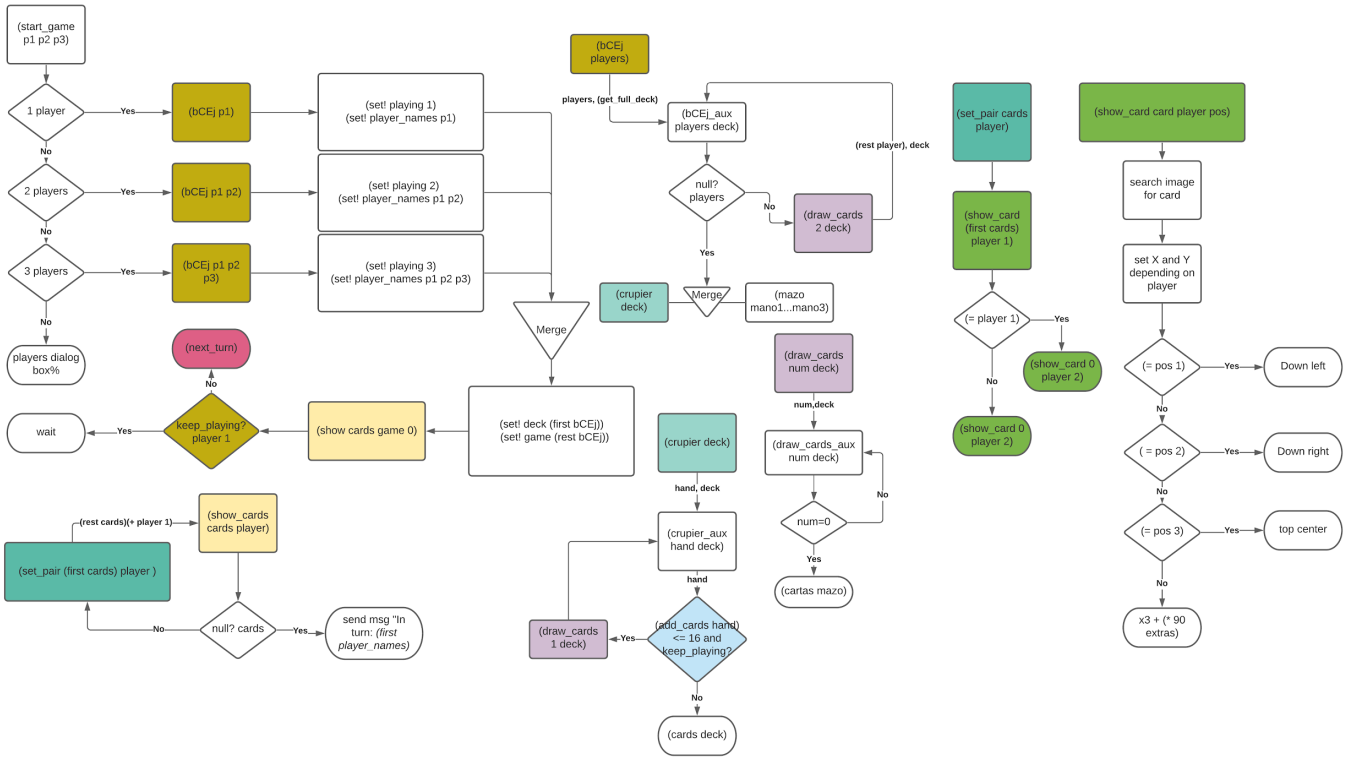
2019165109

Descripción de algoritmos

Algoritmos y Diagramas

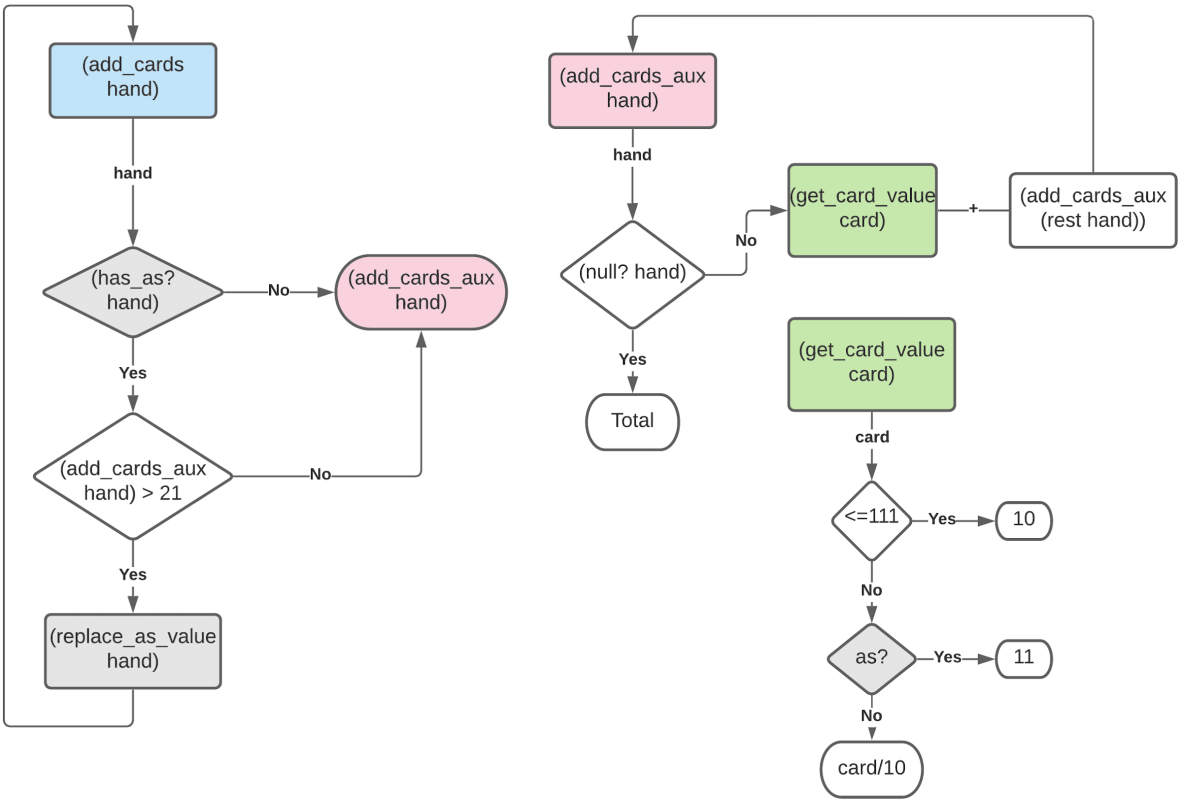
Iniciar el juego y Crupier

Al ejecutar el programa inmediatamente nos solicita los nombres de 3 jugadores. Una vez ingresados se ejecuta la función de GUI (start_game p1 p2 p3) que evalúa la cantidad de jugadores que se han ingresado y ejecuta (bCEj p1 p2 p3) en la lógica. Esta función llama a su auxiliar con una lista conteniendo el mazo completo de cartas, allí, recursivamente ejecuta (draw_cards 2 deck), que extrae 2 cartas aleatoriamente del mazo y las devuelve como una lista junto al nuevo mazo trucado. Este proceso se repite hasta que no haya más jugadores a los que asignar cartas. Allí entonces ejecuta (crupier deck), que agrega cartas recursivamente mediante la función (draw_cards 1 deck) mientras la mano del crupier sume menos o igual a 16, lo que se comprueba mediante la función (add_cards hand). Es decir, la partida completa del crupier se define desde el inicio. Una vez completo se finaliza la función y retorna el mazo truncado junto a las manos de los jugadores. En ese instante continúa la ejecución de (start_game), que actualiza los valores de las variables para el mazo y las manos del crupier y los jugadores. Finaliza utilizando (show cards game 0) que recursivamente muestra la primera carta de cada jugador y la segunda carta boca abajo, salvo para el primer jugador, a quien muestra su segunda carta. Esta función utiliza (set_pair (first cards) player) como auxiliar y está a su vez utiliza (show_card card player pos) que muestra una única carta basada en su identificador, jugador a quien pertenece y posición que debe ocupar en la interfaz gráfica.



Sumar cartas

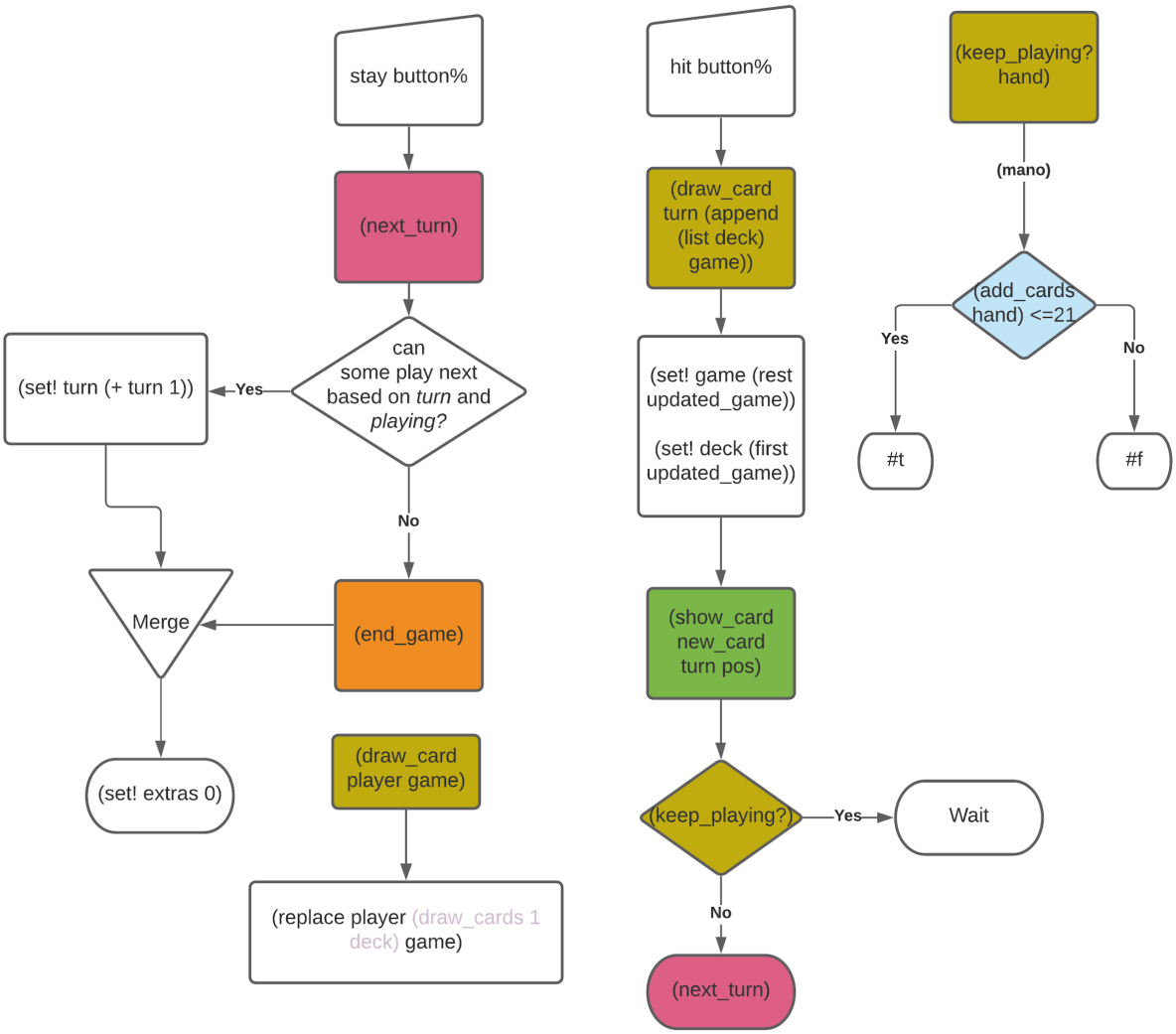
Implementado mediante la función (add_cards hand), esta función recursiva comienza por preguntar si la mano dada tiene un as, mediante (has_as? hand). Si la mano tiene un as , cuyo valor por default es 11, y suma más de 21 se ejecuta (replace_as_value hand), que cambia al as por un número de punto flotante, que hará que su valor sea considerado como 1 al someterse a (integer?). En ese punto se vuelve recursivamente a la función o bien, si no había un as o este no hacía que se sumará más de 21 se ejecuta la función auxiliar. Recursivamente la función suma el valor del primer elemento de la lista con el resto de la lista. Para ello utiliza la función (get_card_value card), donde si la carta es mayor o igual 111, retorna 10, si es un haz 11 y si no, entonces el valor (identificación) de la carta en división entera de 10. El sistema de identificación para las cartas se detalla a más profundidad en la sección de estructuras de datos.



Hit y stay

Al pulsar el botón de hit el proceso para que el jugador tome una carta más inicia. Primero se ejecuta (draw_card turn (append (list deck) game)) lo que ejecuta (replace player (draw_cards 1 deck) game) y devuelve a la interfaz gráfica el juego actualizado. Inmediatamente la interfaz actualiza las variables y ejecuta (show_card new_card turn pos) para mostrar la nueva carta en la interfaz. Seguidamente pregunta si el jugador puede seguir tomando cartas con su nuevo mazo, mediante (keep_playing?), sí si puede, el programa queda a espera de la decisión del jugador. Si no, ejecuta (next_turn), qué exactamente el mismo comando que se ejecuta en el caso que el jugador decida no tomar más cartas y presionar stay. En ese momento la función evalúa si existe algún otro jugador que pueda tener el turno, basándose en la cantidad de jugadores que hay, el turno en el que se estaba y si el jugador ya suma 21. Si hay otro jugador

que pueda continuar actualiza la variable de turno y la de cartas tomadas y queda en espera. Si no existe nadie mas que pueda jugar se ejecuta (end_game) para terminar el juego.



Ganadores y fin del juego

Este algoritmo se ejecuta cuando todos los jugadores han finalizado su turno o no pueden tomar más cartas. Inicia con la función (end_game), esta función lo primero que hace es ejecutar (show_crupier (rest(player_cards 0)) 2), una función recursiva que muestra todas las cartas del croupier en la interfaz salvo la primera, que ya es visible, esta función utiliza a su vez la antes mencionada (show_card card player pos). Una vez todas las cartas del crupier sean visibles la

interfaz actualiza la variable mediante el comando (set! result_strings (winners game)). Aquí se llama a la función lógica (winners game). Su objetivo es determinar el resultado de cada jugador. Compara recursivamente la mano de cada jugador con la del crupier de la siguiente forma. Pregunta si el jugador tiene un blackJack, si lo tiene le otorga esa condición, de lo contrario pregunta si el crupier tiene un blackjack, si lo tiene otorga esa condición, si no fuese así pregunta (lost? crupier player), esta función pregunta si el jugador podría seguir jugando (si tiene menos de 21), si tiene menos verifica con (add_cards player) si suma 21, si no lo suma da false, si si suma 21 pregunta si el crupier tiene menos de 21, al igual que haría si el jugador si tuviera menos de 21 originalmente. Si el crupier tiene más de 21 o 21 da falso, en caso contrario pregunta si la suma de cartas del jugador es menor a la del croupier. Si se cumple entonces devuelve true, de lo contrario es falso. Si (lost? crupier player) devuelve true le da esa condición al jugador, sino pregunta (won? crupier player). Esta función pregunta si el crupier suma más o igual de 21, en caso que lo haga da falso, en caso contrario pregunta si suma 21 exacto. Si suma 21 exacto retorna falso, de lo contrario retorna verdadero. Si (won? crupier player) es verdadero asigna la condición de victoria al jugador, de lo contrario ejecuta (tie? crupier player), que verifica si la suma de las cartas del jugador es igual a la del crupier, en caso de ser verdadero se le asigna la condición de empate al jugador, si no la de victoria. Una vez hecho esto con cada jugador se devuelven los resultados a la interfaz como una lista de strings. La interfaz entonces actualiza los valores en un cuadro de diálogo con los resultados y los muestra al jugador. Si se selecciona “play again”, la interfaz reinicia todas sus variables, incluyendo las imágenes, y muestra el cuadro de diálogo para que los nombres de los nuevos jugadores sean insertados y empezar el todo el proceso una vez más.

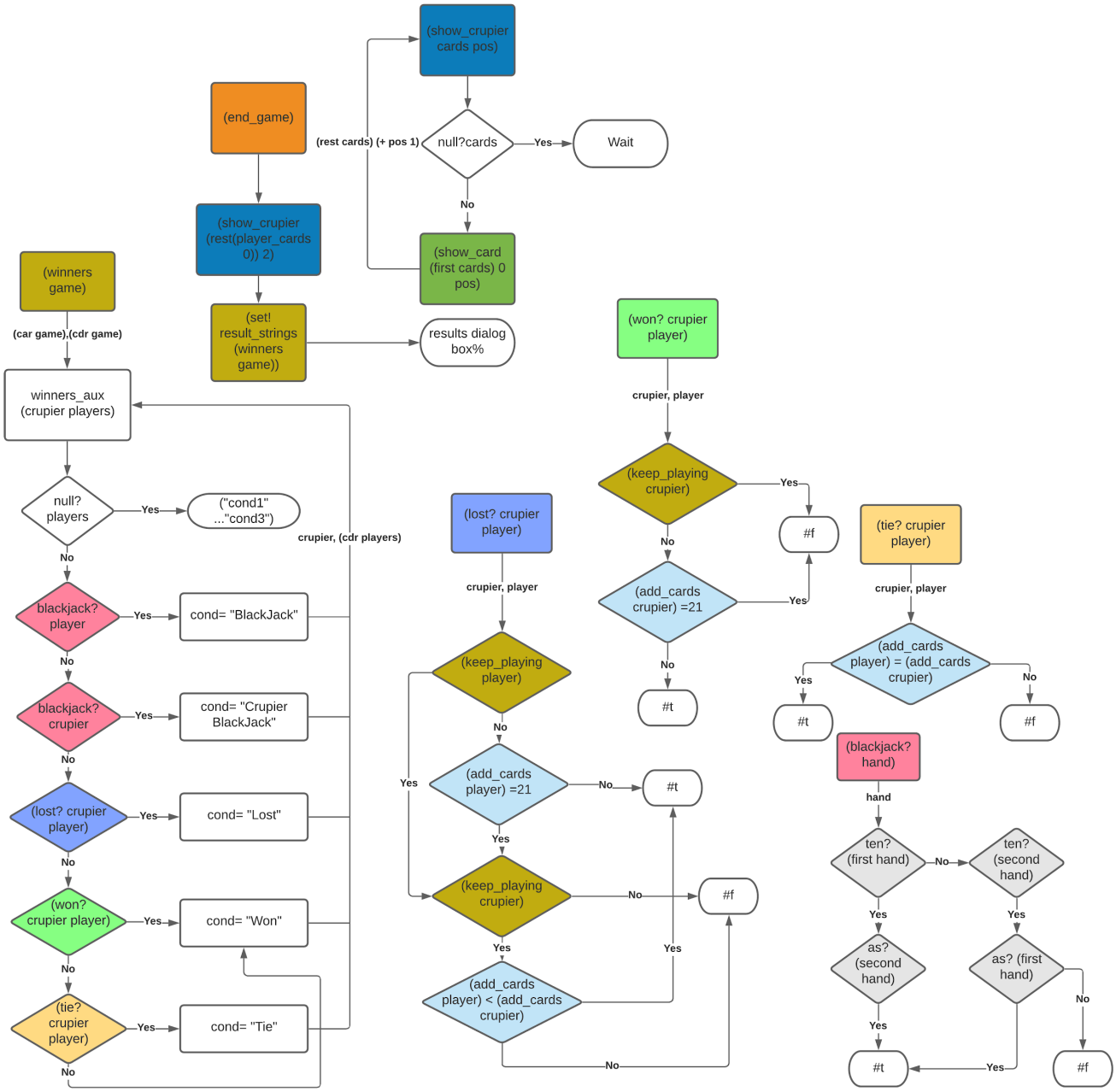
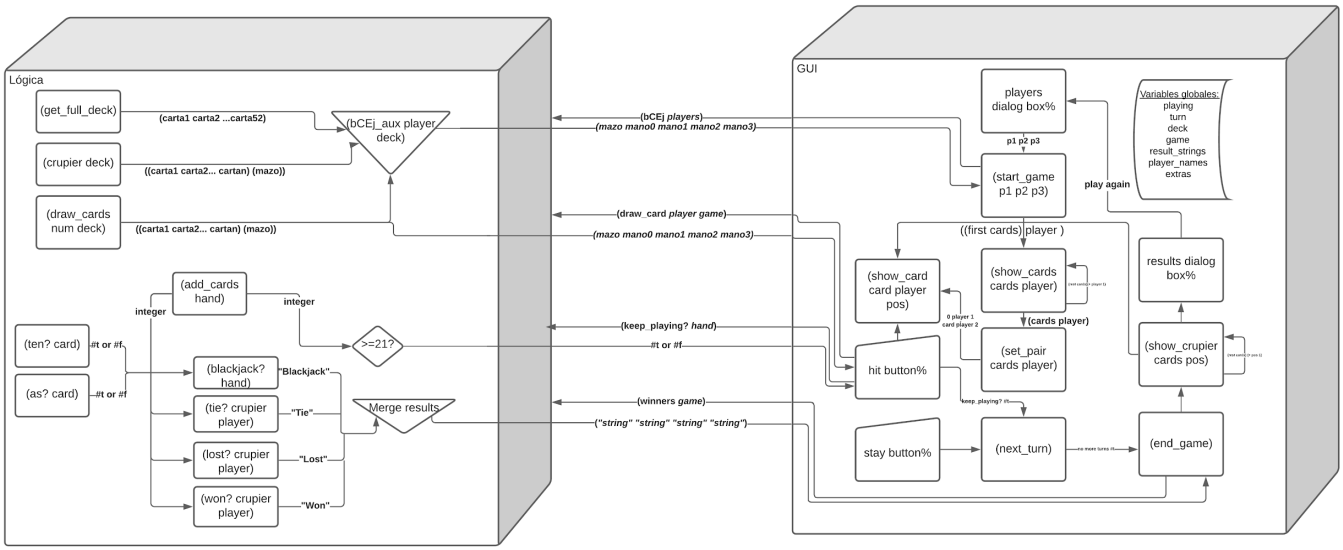

(show_crupier cards pos)
(end_game)
(rest cards) (+ pos 1)
null?cards
Yes
Wait
No
(show_crupier (rest(player_cards 0)) 2)
(show_card (first cards) 0 pos)
(winners game)
(car game),(cdr game)
(set! result_strings (winners game))
results dialog box%
(won? crupier player)
crupier, player
winners_aux (crupier players)
(keep_playing crupier)
Yes
#f
No
(add_cards crupier) =21
Yes
No
#t
null? players
Yes
("cond1" ..."cond3")
crupier, (cdr players)
No
(lost? crupier player)
crupier, player
(keep_playing player)
No
Yes
(add_cards player) =21
No
#t
Yes
(keep_playing crupier)
No
#f
Yes
(add_cards player) < (add_cards crupier)
Yes
No
(tie? crupier player)
crupier, player
(add_cards player) = (add_cards crupier)
Yes
No
#t
#f
blackjack? player
Yes
cond= "BlackJack"
No
blackjack? crupier
Yes
cond= "Crupier BlackJack"
No
(lost? crupier player)
Yes
cond= "Lost"
No
(won? crupier player)
Yes
cond= "Won"
No
(tie? crupier player)
Yes
cond= "Tie"
No
(blackjack? hand)
hand
ten? (first hand)
No
ten? (second hand)
Yes
Yes
as? (second hand)
as? (first hand)
No
Yes
Yes
#t
#f

Diagrama de arquitectura



*Nota: Todos los diagramas se encuentran en formato pdf en el repositorio del código.

Descripción de las funciones implementadas

Logica

bCEj players

Función para inicializar el juego de Blackjack, llama a las funciones crupier para obtener el juego completo del crupier con la lógica respectiva y llama a la función draw_cards para obtener las cartas iniciales de cada jugador.

Input: Una lista de strings con los nombres de los jugadores.

Output: Una lista con: una lista de las cartas, una lista con el juego del crupier y las listas con las 2 cartas iniciales de los jugadores.

Keep_playing?

Función que indica si el jugador puede seguir pidiendo cartas, llama a la función add_cards para conocer el valor total de la mano y determinar si se pasa de 21 para avisar que ya no puede seguir pidiendo cartas.

Input: Una lista con las cartas del jugador.

Output: #t si la suma de las cartas es menor a 21 y #f si es mayor o igual a 21.

draw_card player game

Función para que el jugador pida una carta, al recibir el juego completo y el número de jugador se encarga de encontrar las cartas del jugador especificado y utiliza la función replace para actualizar sus cartas uniendo su mano anterior con la función draw_cards.

Input: El número de jugador que está solicitando una carta y una lista con el juego completo: ((mazo) (cartas del crupier) (cartas J1)...).

Output: Retorna la lista con el juego completo pero le agrega una carta más al jugador especificado y remueve esa carta de la lista del mazo.

winners game

Función que determina la condición final de cada jugador (Ganó, Perdió, Empató, Blackjack, Blackjack del crupier), utiliza las funciones won?, blackjack?, lost? y tie?

Input: Una lista con el juego completo: ((mazo) (cartas del crupier) (cartas J1)...).

Output: Una lista de strings con la condición de cada jugador ("Won" "Tie" "Lost").

get_full_deck

Función que retorna el mazo completo de cartas en orden aleatorio.

Input: No recibe entradas.

Output: Una lista con 52 números enteros que representan todas las cartas del mazo.

lost? crupier player

Función para saber si el jugador perdió. Llama a las funciones keep_playing? Y add_cards.

Input: Una lista con las cartas del crupier y una lista con las cartas del jugador.

Output: #t si el jugador perdió y #f de lo contrario.

Tie?

Función para saber si el jugador empató con el crupier. Llama a la función add_cards para comparar los valores.

Input: Una lista con las cartas del crupier y una lista con las cartas del jugador.

Output: #t si hubo un empate y #f de lo contrario.

Won?

Función para saber si el jugador ganó. Llama a la función add_cards para comparar los valores.

Input: Una lista con las cartas del crupier y una lista con las cartas del jugador.

Output: #t si el jugador ganó y #f de lo contrario.

crupier deck

Función que retorna las cartas del juego completo del crupier. Utiliza la función draw_cards para pedir cartas cuando lo necesita y add_cards para saber si puede pedir una carta más o si se debe quedar plantado.

Input: El deck con el que juega el crupier.

Output:Una lista con: una lista con las cartas del crupier y el mazo sin las cartas del crupier.

draw_cards

Función para sacar cierta cantidad de cartas de un deck.

Input: Un número entero (cantidad de cartas) y una lista con el deck.

Output: Una lista que contiene una lista de las cartas que se sacaron y una lista del mazo sin las cartas.

Replace_as_value hand

Función que reemplaza el valor de un As de 11 a 11.1 que a la hota de obtener el valor va a ser 1 en lugar de 11.

Input: Una lista de cartas.

Output: La misma lista con un As que vale 1.

get_card_value card

Función para determinar el valor numérico de una carta.

Input: El número que representa la carta.

Output: El valor numérico de dicha carta.

as? Card

Función que determina si una carta es un As.

Input: El número que representa la carta.

Output: #t si representa un As con valor de 11 y #f si es otra carta.

Ten?

Función que determina si una carta representa un valor numérico de 10. Utiliza la función get_card_value.

Input: El número que representa la carta.

Output: #t si es una carta representa un valor numérico de 10 y #f si es otra carta.

Blackjack?

Función que determina si las cartas son un Blackjack. Utiliza la función ten? Y as?

Input: Una lista con las cartas.

Output: #t si se formó un Blackjack y #f en el caso contrario.

has_as? Hand

Función que determina si en las cartas hay un As.

Input: Una lista con las cartas.

Output: #t si tiene al menos un As y #f en el caso contrario.

add_cards hand

Función que suma las cartas de una mano. Utiliza las funciones has_as?, replace_as_value y get_card_value.

Input: Una lista con las cartas.

Output: La sumatoria de dichas cartas, en el caso de que contenga un As y se pase de 21, cambia automáticamente su valor.

Interfaz

start_game p1 p2 p3

Comienza el juego. Pide a la lógica que arme la partida inicial. Que corresponde a una lista con el mazo y otra serie de listas con las cartas de los jugadores. También analiza el valor de los strings, si estos están vacíos se considera que no existe tal jugador.

Input: Recibe 3 strings con los nombres de los jugadores.

Output: Llama a show_cards que muestra las cartas iniciales en la interfaz.

show_cards cards player

Función recursiva que muestra los pares iniciales de cartas de los jugadores.

Input: Las cartas de un jugador y un número para referenciarlo.

Output: Llama a set_pair para mostrar los pares de cartas iniciales. Al finalizar el ciclo recursivo muestra el nombre del jugador en turno.

set_pair cards player

Muestra los pares iniciales de cartas de los jugadores. Muestra la primera carta en la primera posición y en la segunda posición muestra la carta boca abajo.

Input: Las cartas de un jugador y un número para referenciarlo.

Output: Llama a show_card para mostrar cada carta inicial.

show_card card player pos

Función que crea el bitmap de la imagen de la carta y lo coloca en su posición en el canvas.

Input: Una carta, un jugador, y la posición donde va la carta (su posición en la lista).

Output: Muestra la carta en la interfaz. Y reproduce el sonido correspondiente.

player_cards player

Retorna las cartas de un jugador o el crupier.

Input: Recibe un número de 0 a 3.

Output: Una lista con las cartas del jugador.

next_turn

Función que se ejecuta cuando se pasa al siguiente turno. Evalúa si el jeugo debe terminar o continuar basándose en la cantidad de jugadores y el turno que seguiría.

Input: No recibe entradas.

Output: Llama a end_game o cambia el turno.

end_game

Función que se llama al verificar que la partida ha terminado.

Input: No recibe entradas.

Output: Termina la partida, muestra las cartas del croupier, modifica los valores y muestra la ventana de resultados y reproduce el sonido correspondiente.

show_crupier cards pos

Función recursiva que llama a show_card para mostrar todas las cartas del crupier.

Input: Una lista con las cartas del crupier menos la primera y la posición donde va la primera carta de la lista.

Output: Mostrar las cartas del crupier.

Estructuras de datos desarrolladas

Inicialmente la lógica envía una única lista a la interfaz. Esta lista en su primer elemento contiene a su vez otra lista, esta segunda lista contiene el mazo completo de la partida menos las cartas iniciales que se han repartido. Excluyendo ese primer elemento la lista se comporta como una matriz (lista de listas) que contiene las manos de cada jugador. Por ejemplo una partida recién iniciada de 2 jugadores se veria asi: ((11 13 54 52 132 22 ….n_{48}) (12 62) (111 83)).

Listas (mazo de cartas)

Para representar el mazo de cartas con el que se está jugando se utiliza una lista con inicialmente 52 números, cada uno representando una carta distinta. Los números varían del 11 hasta el 134. La unidad del número siempre representa el tipo. (1 espada, 2 diamantes, 3 trebol, 4 corazón), mientras que la decena y centena el valor numérico, por lo que por ejemplo todos los 4 del juego se representan así: 41, 42, 43 y 44. Para la J se tiene el valor de 11, Q 12, y K 13, por lo que 132 representaría el rey de diamantes.

Cuando esta lista se envía por primera vez a la interfaz ya se encuentra incompleta pues es enviada junto a las cartas que se han repartido para iniciar la partida. Esta lista es enviada de vuelta a la lógica cada vez que un jugador toma una carta, allí la lógica devuelve la nueva carta y el mazo sin esta carta, allí la interfaz actualiza los valores de las variables y continua el juego.

Lista de listas (manos de los jugadores)

Las cartas que posee cada jugador son representadas por una lista de 2, 3 o 4 listas dependiendo de la cantidad de jugadores. El primer elemento (la primera lista) siempre está reservada para la mano del crupier, la segunda para el primer jugador y así sucesivamente de ser necesario. La posición de cada carta dentro de la lista de cartas de un jugador tiene relevancia, pues en la interfaz se muestran según ese orden. La primera es la carta izquierda, la segunda la

derecha, la tercera la superior y cualquiera más allá de esa se pondrá a una distancia $x_n = x_3 + kn_i$ de la tercera (con k una cantidad de pixeles constante, en este caso 90, y la posición en la lista).

Problemas sin solución

Resolucion dinamica

No se logró encontrar alguna manera para que la ventana fuera escalable. Por esta razón las imágenes tienen una posición y tamaño específicos, y la interfaz es de 1920x1080 pixeles siembre, lo que evita que sea correctamente jugado en monitores más pequeños. Esta decisión fue tomada ya que como no hay un sistema de suma automática (con el fin de mantener la naturaleza del juego original) era necesario ver el número de la carta y con la interfaz original de 720x480 se apreciaba lo suficiente.

Mover imágenes o modificar texto dentro de un canvas

No se buscó profundamente una manera de modificar los elementos de un canvas dinámicamente. Por esta razón las nuevas imágenes solo aparecen sobre las anteriores, visualmente no es notable ni un problema, pero no era la idea original. Sin embargo, cada vez que se reinicia el juego todos los elementos del canvas son eliminados.

Ocultar la “x” en la ventana en el cuadro de diálogo de los jugadores

Con el fin de evitar el error causado al cerrar el cuadro de diálogo donde se ingresa a los jugadores decidió eliminar la opción de poder cerrarlo, sin embargo al utilizar el comando 'no-caption de racket al instanciarlo, cuyo objetivo es eliminar la barra de titulo de windows, no surtió efecto y la barra sigue allí. Por lo que si se cierra la ventana de jugadores, el juego nunca inicia y no hay más remedio que volver a correr el programa y volverlo a correr.

Problemas encontrados

Posición de los elementos en el gui

El kit visual de racket crea los elementos en orden descendente, así que el orden en el código refleja su posición en el frame principal. Es por esta razón, sin embargo, que fue necesario crear un panel horizontal *(horizontal-panel%)* para contener los botones, para que estos se organizaran de manera horizontal y no verticalmente.

Tamaño de los elementos en el gui

Cuando se creó el panel para contener a los botones el tamaño de este interfería con el tamaño del canvas, lo que hacía que la imagen se viera incompleta. Para solucionar esto se utilizaron los comandos *[min-width 1920] [min-height 900]* al crear el canvas. Por otra parte los botones eran muy pequeños respecto a otros elementos en el canvas. Utilizando el mismo comando pero al crear los botones se logra el tamaño deseado.

Texto no aparece completo

Al intentar mostrar un string compuesto por más de una palabra la clase *message%* ubicada sobre el canvas no lo mostraba a completitud, este mensaje se utiliza para mostrar el nombre del jugador en turno y mensajes de bienvenida y agradecimiento. Fue hasta que se agrego el comando *[auto-resize #t]*.

Dar vuelta a las cartas

La idea original consistía en tomar el bitmap del elemento en el canvas e intercambiarlo por otra imagen. Dada la simplicidad del programa, y que la cantidad de elementos máximos es relativamente poca, se decidió superponer la imagen de la carta sobre la imagen de la carta cabeza abajo para simplificar el proceso.

Encontrar la ruta de los archivos del audio, windows y linux.

Por alguna razón que se desconoce los archivos de audio no pueden ser referenciados como las imágenes con tan solo la ruta relativa al programa donde se encuentran. Fue necesario dar la ruta completa del archivo de sonido, para ello se utilizó *(path->string (current-directory))* que sería el equivalente a la ruta donde está el archivo ejecutándose expresada como string. Con ella solo es necesario. Ya que la sintaxis de rutas es distinta de sistema a sistema, en linux es necesario usar el carácter "/" en vez de "\\", sin embargo windows no tiene problema en utilizar "/" aun cuando el resto de la ruta use "\\". Es decir, "*carpeta\\subcarpeta/archivo"* Es una ruta válida en windows, pero no en linux.

Identificar tipo y valor de cada carta

Al elegir como se iba a representar el deck se presentó un problema al no considerar en la lógica el tipo de carta además de su valor porque la interfaz necesitar saber cual carta mostrar exactamente, se resolvió identificando las cartas con números donde las unidades son números del 1 al 4 que representan al tipo y las decenas y centenas representan el número de la carta.

Controlar el valor del as

El valor del As fue un problema al crear la función keep_playing? Ya que en muchos casos el valor era mayor a 21 pero el jugador podía seguir jugando porque tenía un as en sus cartas, para esto se crearon las funciones has_as? Para identificar si el jugador tenía un as un su mano y change_as_value para cambiar el valor del primer as del mazo que en este caso un as que vale 1 se representa como 11.1.

Determinación de condiciones finales

Inicialmente para determinar si los jugadores ganan, pierden o empatan con respecto al crupier había una función llamada winners que provocaba algunos errores, por ejemplo cuando un jugador obtenía un mazo de cartas con un valor de 21 que no era un blackjack, al preguntar si

podía seguir jugando con la función keep_playing? Decía que no y el jugador perdía automáticamente. Habían pequeños errores de ese tipo y se resolvió con una mejor modularización y las funciones lost?, won? y tie?.

Plan de actividades

Descripción de la tarea	Tiempo estimado de completitud	Responsable a cargo	Fecha de Entrega
Crear una función para iniciar el juego	1h	Natalia	11/03/21
Crear una función para escoger cartas de un mazo aleatoriamente	1h	Natalia	10/03/21
Crear una función para repartir los mazos iniciales (Incluye creación de la estructura de datos)	2h	Natalia	11/03/21
Función que maneje la lógica del crupier	3h	Natalia	11/03/21
Crear una función para validar si las cartas del jugador forman un Blackjack	1h	Natalia	11/03/21
Crear una función en caso de que el jugador pida otra carta	1h	Natalia	11/03/21
Crear una función en caso de que el jugador se plante	1h	Natalia	11/03/21
Crear una función que valide si el jugador se pasa de 21 para terminar su turno automáticamente	1h	Natalia	12/03/21

Crear una función que indique el ganador (cada jugador o el crupier)	1h	Natalia	15/03/21
Crear una funcion que inicializa la interfaz gráfica (Incluye la investigación)	3h	José Ale	9/03/21
Crear una función que actualiza el GUI si un jugador pide una carta	1h	José Ale	11/03/21
Crear una función que actualiza el GUI si un jugador se planta	1h	José Ale	12/03/21
Confeccionar u obtener las imágenes necesarias para el GUI	2h	Jose Ale	12/03/21
Crear una función que actualiza el GUI con los ganadores	1h	José Ale	14/03/21

Conclusiones

- La creación de funciones modulares simples puede mejorar la legibilidad del código y por ende la facilidad de revisión y mantenimiento.
- La modularización del apartado lógico y gráfico permite trabajar con relativa libertad sí que definen una serie de funciones mediante las cuales se establecerá su comunicación.
- La documentación oficial del kit gráfico de racket ofrece una amplia guía para la utilización de sus herramientas.
- Es posible representar mazos y manos de cartas utilizando listas y operaciones sobre las mismas.
- La función shuffle sirve para desordenar una lista aleatoriamente, en este caso el mazo de cartas.
- Se puede utilizar recursividad para crear funciones muy específicas para el tratamiento de listas.
- Existen funciones preestablecidas como list-ref para tomar un elemento de la lista a partir de su índice.
- Los diagramas de flujo son una forma muy clara de representar una función. Lo que facilita su comprensión y explicación.
- Analizar a profundidad el problema y las tareas que lo componen ayuda a un mejor planteo de la solución y equilibrio del trabajo.
- Trabajar con una ventana de tamaño fijo permite agregar simpleza al código, pero puede restarle comodidad al usuario.
- El bitmap% de racket resta calidad a la imagen que se utiliza.
- El panel% permite acomodar a sus hijos horizontal o verticalmente.

Recomendaciones

- Crear funciones modulares para mejorar la legibilidad del código.
- Modularizar el código para trabajar con mayor grado de libertad. Esto pues es un grupo pequeño.
- Definir claramente las interacciones entre módulos del programa. Esto ayuda a evitar contratiempos e incongruencias al acoplar los distintos módulos de un programa.
- Tener a disposición la documentación de cualquier kit de software que se vaya a utilizar. Leer con detenimiento y atención puede ayudar a un mejor uso del kit, y agilizar la resolución de problemas.
- Considerar diversas estructuras de datos para manejar y manipular información.
- Realizar diagramas para visualizar la solución entera o tan solo funciones. Esto es una excelente guía para comprender a cabalidad la solución descrita.
- Analizar profundamente el problema. Esto ayuda a encontrar los mayores retos y comprender plenamente la solución a la que se desea llegar.
- Analizar si una resolución dinámica es verdaderamente necesaria, pues de no serlo trabajar con un tamaño de ventana único simplifica la solución y agiliza el proceso. Es recomendable escoger, en la medida de lo posible, una resolución que sea cómoda para la mayoría de los usuarios.
- Si se desea organizar los elementos en la interfaz de manera vertical u horizontal se puede utilizar la clase panel%.
- La creación de funciones simples de output booleano pueden ayudar a mejorar la lectura de código.
- Si se necesita una función que implique el tratamiento de listas se puede utilizar recursividad para poder recorrer cada elemento y a su vez crear el resultado deseado.
- Tener claras las diferencias que pueden existir al trabajar en diferentes S.O, en este caso las rutas de los archivos, para poder adaptar el programa a cada uno.

Bibliografía

[1] M.Flatt, R.B.Findler, J.Clements “The Racket Graphical Interface Toolkit”. *Racket Docs.* 2017 [Online]. https://download.racket-lang.org/releases/6.9/doc/gui/index.html [Accesado: Mar 9, 2021]

[2] Dyoo. “Insert an image into frame x y coordinates”. *StackOverflow*. 2013. [Online]. https://stackoverflow.com/questions/16929093/insert-an-image-into-a-frame-x-y-coordinates [Accesado: Mar 9, 2021]

[3] M.Flatt, R.B.Findler, J.Clements. “The Racket Drawing Toolkit”. *Racket Docs* . 2017 [Online]. https://docs.racket-lang.org/draw/bitmap_.html [Accesado: Mar 9, 2021]

[4] M.Flatt, R.B.Findler, J.Clements. “Module Basics”. *Racket Docs.* 2017 [Online]. https://docs.racket-lang.org/guide/module-basics.html[Accesado: Mar 9, 2021]

[5] NaturalComedy. ”How to play sounds in beginner student?”. *Reddit* [Online]. https://www.reddit.com/r/Racket/comments/2w8omx/how_to_play_sounds_in_beginner_student/ [Accesado: Mar 14, 2021]

[6] M.Flatt, R.B.Findler, J.Clements. “Manipulating Paths”. *Racket Docs.* 2017 [Online]. https://docs.racket-lang.org/reference/Manipulating_Paths.html#%28def._%28%28quote._~23~25kernel%29._complete-path~3f%29%29 [Accesado: Mar 14, 2021]

[7] J.E. Guzman. . “Introducción a la programación con Scheme.” *Cartago: Editorial Tecnológica de Costa Rica.* 2006.