**Instituto Tecnológico de Costa Rica**

**Área de Ingeniería en Computadores**

CE-3104: Lenguajes, Compiladores e Intérpretes

**Profesor:**

Marco Rivera Meneses

**Tarea 1: BlaCEjack**

Paradigma funcional

**Integrantes:**

Carlos Adrian Araya Ramirez

Michael Shakime Richards Sparks

José Andrés Solano Mora

**Primer semester 2021**

1. **Breve descripción del proyecto**

BlaCEjack se es un juego basado en el juego de casino llamado Black Jack o también comúnmente llamado 21. Consiste en obtener una mejor puntuación que el crupier a partir de pedir cartas y aproximarse lo más posible a una puntuación de 21 sin pasarse.

* 1. **Descripción detallada de los algoritmos de solución desarrollados con sus respectivos diagramas.**

La acción o función principal de este juego consiste en repartir una carta aleatoria dentro de un mazo de 52 cartas diferentes a un jugador en momento dado, y analizar la condición de dicho jugador a partir de las cartas que posee.

En base a esta función, el flujo del programa consiste en primero seleccionar el numero de jugadores, según la cantidad de jugadores seleccionada se creará una lista de jugadores que funcionará de estructura de datos durante todo el juego.

Una vez creada la lista de jugadores, el algoritmo inmediatamente revuelve la cartas del mazo (hace un shuffle a la lista de cartas) y reparte dos cartas a cada jugador y al crupier, donde la primera carta repartida equivale a la carta boca abajo. Durante el reparto de estas primeras dos cartas no se hace ningún calculo sobre la puntuación de ningún jugador, solo se hace un cdr de la lista de cartas para evitar que se repitan.

Cuando todos los jugadores y el crupier tienen las primeras dos cartas comienza el primer turno, donde queda a la espera de que el jugador indique si desea solicitar una carta más o plantarse, si el jugador decide plantarse continuará el turno del siguiente jugador y si decide pedir una carta se evaluarán las siguientes condiciones.

* **Si la puntuación del jugador con las cartas visibles para todos es mayor a 21,** el jugador habrá perdido y continua el turno del siguiente jugador.
* **Si la puntuación del jugador con las cartas visibles para todos es menor a 21,** se le da una carta al jugador, es decir se agrega a su lista de cartas el car del mazo y se le hace un cdr al mazo para sacar la carta repartida.
* **Si es el turno del crupier,** se evalúan las mismas dos condiciones anteriores pero esta vez si se toma en cuenta la carta volteada.

El proceso de pedir una carta de se repite hasta que todos los jugadores se hayan plantado o hayan perdido y llegue el turno del crupier.

El turno del crupier es el turno final, por lo tanto, a partir de aquí cada vez que el crupier pide una carta, se analiza si obtiene un blackjack ya que en este caso ganaría irrefutablemente.

Finalmente, cuando el crupier decide plantarse o pierde, se hace un análisis de los puntajes de todos los jugadores y el crupier y se calcula quien es el ganador.

* 1. **Descripción de las funciones implementadas.**

1. (bCEj X)

- Crea la estructura de los jugadores con la cantidad de jugadores pasada por parámetro.

- Entradas: X: cantidad de jugadores (de 1 a 3)

- Salidas: lista de jugadores con sus valores en cero.

1. (dar-carta jugador mazo)

-Da una carta al jugador pasado por parámetro y devuelve ese mismo jugador modificado con su nueva carta.

-Entradas: jugador de la forma: ("nombre" puntuación '(lista cartas) )

-Salida: el mismo jugador con la nueva carta: ("nombre" puntuación '(lista cartas + carta nueva))

1. (dar-carta-jugador jugador listaJugadores mazo)

-Da una carta a un jugador y devuelve la lista con todos los jugadores y el jugador con la carta nueva actualizado.

-Entradas: jugador: nombre del jugador, listaJugadores: lista de jugadores, mazo: la lista de cartas de donde se dará la carta al jugador

-Salida: lista con todos los jugadores y el jugador con la carta nueva actualizado.

1. (reparte-cartas listaJugadores mazo)

- Reparte una carta a cada jugador y devuelve la lista con todos los jugadores actualizada con su carta nueva cada uno.

- Entradas: listaJugadores: lista de jugadores, mazo: lista de cartas para repartir

- Salida: lista de jugadores actualizada con una carta nueva para cada jugador.

1. Funciones de shakime
2. (turno-crupier crupierDeck mazo)

- Analiza si la puntuación del crupier es menor a 16 para solicitar otra carta o de lo contrario plantarse.

- Entradas: crupierDeck: estructura del crupier, mazo: lista de cartas para repartir.

- Salidas: crupier actualizado

* 1. **Descripción de la ejemplificación de las estructuras de datos desarrolladas.**

La estructura de datos de los jugadores es una lista de jugadores, donde cada jugador es una lista, en la que su primer elemento (el car) es el nombre del jugador, el segundo elemento, es la puntuación actual equivalente a sus cartas, y su tercer y ultimo elemento es la lista de cartas.

**'(("Player1" 0 ((A 1) (A 2) (A 7))) ("Player2" 0 ((A 6) (A 5) (T 1))) ("Player3" 0 ((A 1) (A 7))))**

La lista de cartas de cada jugador sigue la regla de las cartas que se utiliza en el mazo del juego, donde cada carta es un par ordenado, donde el primer elemento es el palo de la carta y el segundo el valor de la carta.

* 1. **Problemas sin solución: En esta sección se detalla cualquier problema que no se ha podido solucionar en el trabajo.**
  2. **Problemas encontrados: descripción detallada, intentos de solución sin éxito, soluciones encontradas con su descripción detallada, recomendaciones, conclusiones y bibliografía consultada para este problema específico.**
  3. **Plan de Actividades realizadas por estudiante: Planeamiento de las actividades, descripción de la tarea, tiempo estimado de completitud, responsable a cargo y fecha de entrega.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tarea** | **Tiempo estimado** | **Responsable** | **Fecha de entrega** |
| Crear estructura de datos para las cartas. | 1h | José Solano | 09/03/2021 |
| Crear estructura de datos para los jugadores. | 1h | Adrián Araya | 09/03/2021 |
| Implementar la función (bCEj X). | 1h | Shakime Richards | 10/03/2021 |
| Implementar una función para repartir una carta a un jugador. | 2h | Adrián Araya | 11/03/2021 |
| Implementar la función repartir carta al crupier. | 2h | Adrián Araya | 12/03/2021 |
| Investigar como importar funciones de otro archivo en Scheme. | 1h | José Solano | 12/03/2021 |
| Calcular el puntaje de las cartas boca arriba de un jugador. | 2h | Shakime Richards | 12/03/2021 |
| Calcular el puntaje total de todas las cartas. | 1h | Shakime Richards | 13/03/2021 |
| Investigar como guardar valores a variables definidas en Scheme para la interfaz. | 1h | Adrián Araya | 13/03/2021 |
| Crear las funciones que conectarán la lógica con la interfaz. | 3h | Adrián Araya | 13/03/2021 |
| Investigar sobre interfaces graficas en Scheme | 1h | José Solano | 13/03/2021 |
| Crear la función turno que decidirá si debe repartir a un jugador, al crupier o definir el ganador | 3h | José Solano | 13/03/2021 |
| Conectar las funciones de los jugadores con la interfaz | 3h | Adrián Araya | 14/03/2021 |
| Conectar las funciones del crupier con la interfaz | 3h | Shakime Richards | 14/03/202 |

* 1. **Conclusiones.**

1. Aplicando el paradigma de la programación funcional se logró crear una versión del popular juego Black Jack utilizando el lenguaje de programación Racket.
2. Se aplicaron únicamente los conceptos del paradigma de programación funcional, haciendo uso únicamente de funciones recursivas, utilizando listas como estructura de datos principal y sin definir variables.
3. El problema fue resuelto utilizando estructuras de datos basadas en listas, donde estas listas poseen como elementos la información necesaria para darle continuidad al juego, utilizando funciones auxiliares para editar y actualizar estas listas basadas principalmente en el uso de las operaciones primitivas “car” y “cdr”.
4. En base al código final se puede concluir que los problemas resueltos utilizando programación funcional son notablemente más cortos que en otros paradigmas.
   1. **Recomendaciones.**
   2. **Bibliografía consultada en todo el proyecto**