

Instituto Tecnológico de Costa Rica

Centro Académico de Alajuela

IC-4700: Lenguajes De Programación



Tarea Programada 01:

Juego Shisima en Racket

Estudiantes:

William Gerardo Alfaro Quirós – 2022437996

Jesús Gabriel Cordero Diaz - 2020081049

Profesora:

María Auxiliadora Mora Cross

Fecha de entrega:

08/09/23

II Semestre

Descripción del problema

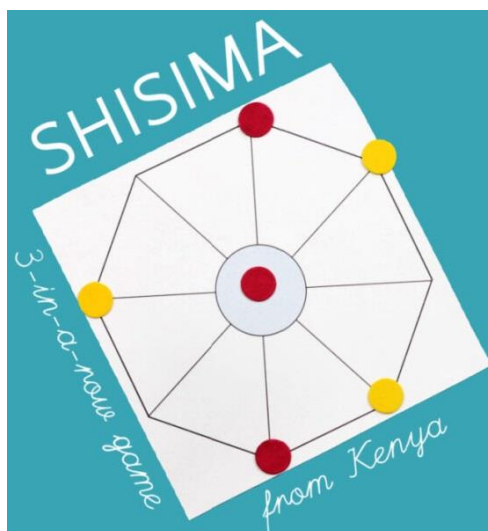
La presente tarea programada consiste en diseñar e implementar en el lenguaje de programación Racket un programa que permita al usuario jugar el juego Shisima mediante una interfaz gráfica y comandos por terminal. Shisima es un juego tradicional de estrategia similar al tres en línea, originario de Kenia, que se juega con fichas móviles en lugar de papel y lápiz. En Kenia, el término "Shisima" significa cuerpo de agua y está representado por el tablero de juego, el cual es un octágono dividido por 4 líneas. Las fichas del juego se llaman "imbalavalia", que significa insectos acuáticos, y estos insectos acuáticos compiten a lo largo del tablero hacia el agua (centro del tablero). Este "puzzle" competitivo añade un giro nuevo al mundo de los juegos de tres en línea con su tablero octogonal. Para ganar en Shisima, los jugadores deben implementar una estrategia eficaz y mostrar disposición para resolver problemas.

Se pretende que cada jugador tenga la experiencia de un juego de mesa competitivo, emulando mediante el uso del teclado selecciones interactivas para realizar movimientos de fichas. El juego al tener una forma octagonal (ver imagen 1), permite distintas combinaciones al jugador; sin embargo, este debe de tener ciertas consideraciones en cuenta al momento de su turno y al ejecutar su movimiento. De esta forma, se espera que el jugador implemente una estrategia en función de ganarle al oponente. En cuanto a la jugabilidad, cada jugador seleccionará mediante un acuerdo quién será el jugador que iniciará la partida, y ambos tendrían tres fichas de color amarillo o rojo, y a su vez, tendría un turno cada uno sucesivamente tomando en cuenta las siguientes restricciones:

- Solo puede mover fichas de su mismo color
- Debe de realizar solo un movimiento por turno
- Los movimientos deben ser únicamente adyacentes
- Mover a la posición central es permitido
- Las fichas únicamente se pueden mover a espacios disponibles (gris)
- Cada jugador dispone de un turno efectivo para realizar su movimiento
- Si el movimiento no es válido, el jugador deberá mover de nuevo
- No se permite brincar ninguna otra ficha al realizar el movimiento
- No hay tiempo límite para realizar cada movimiento

Imagen 1

Forma octagonal del tablero



Nota. Tablero de Shisima

Diseño de componentes

Para la implementación del juego se tomaron en cuenta los siguientes requerimientos provistos por la consigna de la tarea programada:

- La tarea debe programarse en Racket (la tarea no será aceptada en otro lenguaje de programación).
- En el código: no pueden usar iteraciones, por ejemplo, pero no limitadas a “for”, “for*” o “for/list”. Pueden usar variables globales, pero no más de 2 (sin tomar en cuenta las variables requeridas por las bibliotecas de manejo de la interfaz gráfica).
- El sistema debe implementar funcionalidad en un ambiente gráfico agradable utilizando alguna de las bibliotecas para manejo de interfaz gráfica disponibles en Racket.

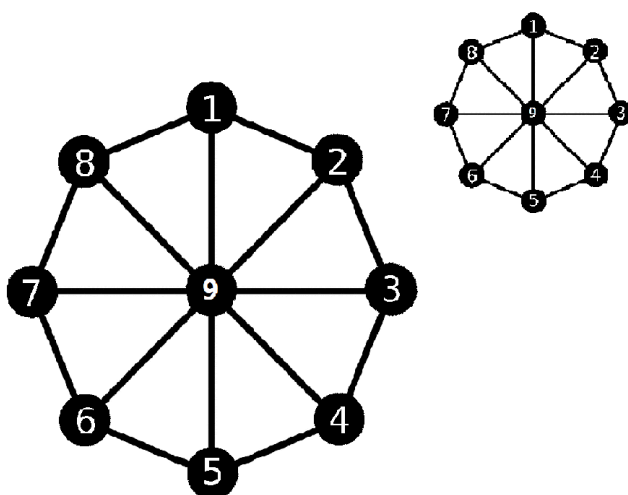
Tomando en cuenta las consideraciones anteriores, se decidió diseñar primeramente la lógica principal del juego y sus funciones principales. Consecuentemente, se crearon dos structs para utilizar *nodo* (en este caso las fichas) y uno para *ficha-turno* que implementa el origen y el destino de la ficha, además de contener la condición de parada del programa *fin*. Para estos structs se debió tomar en cuenta que debían ser mutables y transparentes para estos sean

accesibles tanto como funcionales, y adicionalmente puedan ser modificables después de su creación. Además, una de las mayores limitantes de la tarea programada fue el no poderse usar más de una variable global. Esto responde a dos motivos esencialmente; por un lado, pueden causar efectos secundarios inesperados (colaterales); es decir, dado que las variables globales pueden ser accedidas y modificadas desde cualquier parte del código, es más probable que se produzcan efectos secundarios no deseados. Un cambio en el valor de una variable global en un lugar del código puede afectar inesperadamente el comportamiento en otro lugar. Por el otro lado, hacen que el código sea menos modular. Las variables globales pueden romper la encapsulación y modularidad del código. Los módulos o funciones que dependen de variables globales pueden ser menos reutilizables y más difíciles de probar de forma aislada. Por lo tanto, se decidió solamente utilizar una variable global que tuviera a la ficha del turno actual y se le asigna el valor que sea el resultado de la llamada a la función *make-ficha-turno*.

Para hacer el juego jugable mediante una interfaz gráfica agradable, se utilizaron los módulos de Racket para lectura de imágenes. De esta forma, se diseñó el tablero octagonal donde los jugadores podrían comenzar su partida. Esta imagen fue diseñada y almacenada en formato .png y se rutea mediante la librería de Racket implementando el *bitmap/file* como se muestra a continuación:

Imagen 2

Fondo utilizado para la interfaz del tablero



Nota. Imagen de autoría propia

En cuanto a las funciones utilizadas, se crearon las siguientes:

- **(*modificar-contenido mundo id nuevo-color*)**: se utiliza para cambiar el color de un nodo en una lista de nodos llamada "mundo". Esta función toma tres entradas: la lista de nodos (*mundo*), el número de identificación del nodo que se debe modificar (*id*), y el nuevo color que se desea asignar (*nuevo-color*). La función recorre la lista de nodos y, cuando encuentra el nodo con el *id* especificado, reemplaza su color con el *nuevo-color*. Luego, devuelve una nueva lista de nodos con la modificación realizada. Si no se encuentra el nodo con el *id* especificado, la función devuelve la lista original sin cambios.
- **(*modificar-contenido mundo id nuevo-color*)**: se utiliza para obtener el color de un nodo en una lista de nodos llamada "mundo" basándose en su identificador (*id*). Esta función recibe dos entradas: la lista de nodos (*mundo*) y el número de identificación del nodo (*id*). La función recorre la lista de nodos y, cuando encuentra el nodo con el *id* especificado, devuelve el color de ese nodo. Si no se encuentra un nodo con ese *id*, la función devuelve la lista mundo tal como está. En resumen, esta función permite buscar y recuperar el color de un nodo en función de su *id* dentro de una lista de nodos.
- **(*comparar-cadenas cadena1 cadena2*)**: se utiliza para comparar dos cadenas de texto (*cadena1* y *cadena2*) y determinar si son idénticas. Esta función devuelve **#t** si las cadenas son iguales, es decir, si contienen exactamente los mismos caracteres en el mismo orden. En caso contrario, devuelve **#f**. Es una función que facilita la comparación de cadenas en un programa.
- **(*mover-ficha-aux id-nodo-mover id-nodo-destino mundo*)**: esta función auxiliar es utilizada para verificar si un movimiento de ficha es válido. Recibe tres entradas: el número del nodo a mover (*id-nodo-mover*), el número del nodo de destino (*id-nodo-destino*), y una lista de nodos que representa el "mundo". La función verifica si el movimiento es válido según una serie de condiciones que involucran la comparación de colores en nodos específicos y posiciones en el juego. Si el movimiento cumple con las condiciones, la función devuelve **#t**, indicando que el movimiento es válido; de lo contrario, devuelve **#f**, indicando que el movimiento no es válido.
- **(*mover-ficha id-nodo-mover id-nodo-destino mundo*)**: la función mover-ficha se utiliza para mover una ficha de un nodo a otro en un juego representado por una lista de nodos

llamada "mundo". Recibe tres entradas: el número del nodo a mover (*id-nodo-mover*), el número del nodo de destino (*id-nodo-destino*), y la lista de nodos que representa el estado del juego. La función verifica si el movimiento es válido y actualiza la lista de nodos en función de las reglas del juego. Si el movimiento es válido, se actualiza la lista de nodos con la ficha movida y se devuelve la nueva lista de nodos. Si el movimiento no es válido, se muestra un mensaje de "movimiento no permitido" y se devuelve la lista de nodos original sin cambios.

- **(finalizar-juego color-nodo-ganador):** se utiliza para finalizar un juego y mostrar al ganador. Recibe una entrada, que es el color del nodo ganador (*color-nodo-ganador*). La función muestra un mensaje de felicitaciones ("Felicidades ganaste") y luego utiliza una estructura condicional (*cond*) para determinar cuál de los dos colores (rojo o amarillo) ganó el juego, basándose en la comparación del color del nodo ganador con "red". Si el color del nodo ganador es "red", muestra un mensaje que dice "Ganaron las fichas color Rojo", de lo contrario, muestra "Ganaron las fichas color Amarillo".
- **(validar-gane mundo):** se utiliza para validar si se ha alcanzado un estado de ganador en el juego, basándose en el color de los nodos en una lista de nodos llamada "*mundo*". Esta función no modifica el "*mundo*" original, pero puede devolver una nueva lista de nodos si se cumple una condición de ganador. La función primero obtiene el color de los nodos en posiciones específicas dentro del "*mundo*" utilizando la función *obtener-color-por-id*. Luego, utiliza una serie de condiciones (condiciones de ganador) para verificar si se cumple una de las siguientes condiciones y si ninguna de estas condiciones de ganador se cumple, la función devuelve el "*mundo*" original sin cambios:
 - Si los colores en las posiciones 1, 5 y 9 son iguales y no son "grey", se llama a la función *finalizar-juego* con el color del nodo en la posición 9 como ganador.
 - Si los colores en las posiciones 2, 6 y 9 son iguales y no son "grey", se llama a la función *finalizar-juego* con el color del nodo en la posición 9 como ganador.
 - Si los colores en las posiciones 3, 7 y 9 son iguales y no son "grey", se llama a la función *finalizar-juego* con el color del nodo en la posición 9 como ganador.
 - Si los colores en las posiciones 4, 8 y 9 son iguales y no son "grey", se llama a la función *finalizar-juego* con el color del nodo en la posición 9 como ganador.

- **(pintar-mundo mundo):** se utiliza para representar gráficamente el estado del *mundo* de un juego en la pantalla. Esta función toma una entrada, que es una lista de nodos llamada "*mundo*". La función utiliza la función *place-images* para colocar imágenes de fichas en posiciones específicas en la pantalla, representando así el estado del juego. Cada ficha se representa como un círculo sólido con un radio de 40 unidades y un color obtenido a partir de la lista de nodos "*mundo*" utilizando la función *obtener-color-por-id*. Las coordenadas de posición de cada ficha se especifican en relación con el tablero de juego. Esta función a grandes rasgos, se encarga de pintar gráficamente el estado del mundo del juego en la pantalla, utilizando imágenes de fichas con colores correspondientes a la configuración del juego en la lista de nodos "*mundo*".
- **(seleccionar-ficha mundo key):** se utiliza para gestionar la selección de fichas y realizar movimientos en un juego. Primero, verifica si se ha alcanzado un estado de ganador en el juego y finaliza el juego si es necesario. Luego, según la tecla presionada, permite reiniciar el juego, seleccionar una ficha como origen y un destino válido, y realizar movimientos en función de estas selecciones. La función también continúa verificando si se ha alcanzado un estado de ganador después de cada movimiento y reinicia la información de la ficha seleccionada al final.
- **(mundo-inicio):** se utiliza para inicializar el mundo del juego con una configuración inicial predeterminada. Esta función no toma entradas y devuelve una lista de nodos que representa la configuración inicial del juego. En esta configuración inicial, algunos nodos tienen el color "*yellow*", algunos tienen el color "*red*", y otros tienen el color "*grey*" (vacío). Esta lista de nodos representa el estado inicial del juego antes de que los jugadores realicen movimientos.
- **(big-bang):** se utiliza para configurar y ejecutar un bucle principal para el juego, donde se gestiona la interacción del usuario, se actualiza la representación gráfica del mundo y se verifica si se cumple una condición de finalización. Es una estructura comúnmente utilizada en programación de juegos y simulaciones en Racket para crear aplicaciones interactivas.

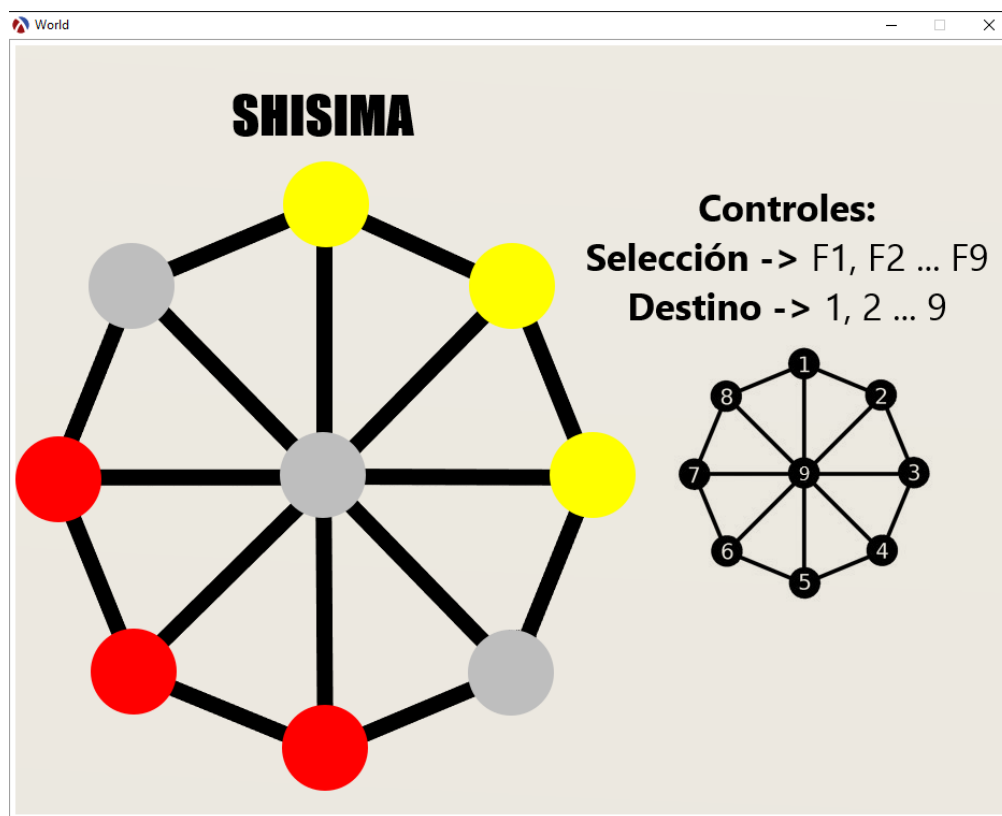
Ejecución del juego

Para jugar al Shisima, se debió tomar en cuenta que dos jugadores jugarían localmente en un mismo ordenador. Consecuentemente, el diseño de la interfaz gráfica está pensado en la practicidad y dinámica que puede involucrar este juego de forma análoga a como lo haría un juego de mesa con tablero físico.

A pesar de este juego ser muy intuitivo, los jugadores podrían tener dudas en cuanto a las reglas generales del *Tres en Línea*; por lo tanto, se toman dos componentes principales para la ejecución del juego: la interfaz gráfica (ver imagen 3) que pretende ser tan explícita como las reglas y las instrucciones que despliega la terminal del IDE (ver imagen 4) como se muestra a continuación:

Imagen 3

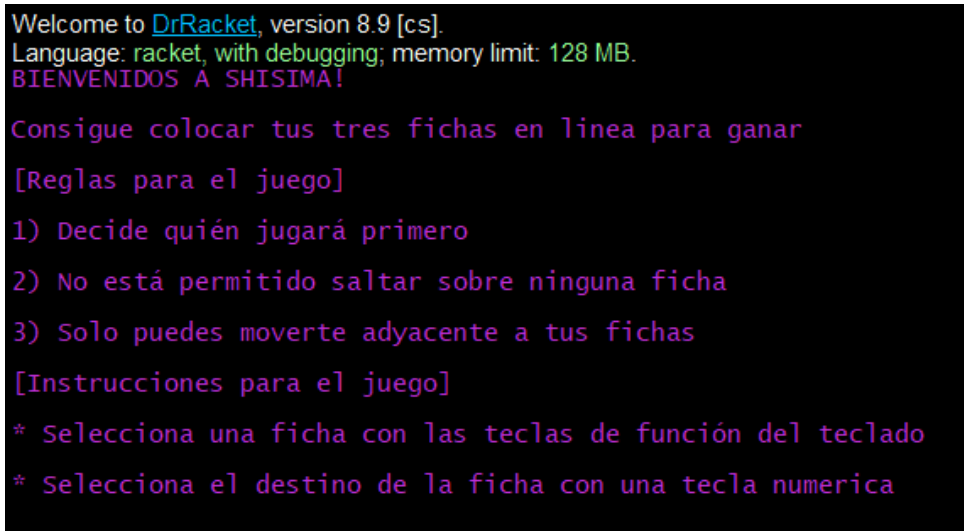
Interfaz gráfica del juego Shisima



Nota. Las tonalidades de la paleta de colores y el tamaño de la ventana podría cambiar según el hardware en el que se corre el programa.

Imagen 4

Terminal del IDE DrRacket



```
Welcome to DrRacket, version 8.9 [cs].  
Language: racket, with debugging; memory limit: 128 MB.  
BIENVENIDOS A SHISIMA!  
  
Consigue colocar tus tres fichas en línea para ganar  
[Reglas para el juego]  
1) Decide quién jugará primero  
2) No está permitido saltar sobre ninguna ficha  
3) Solo puedes moverte adyacente a tus fichas  
[Instrucciones para el juego]  
* Selecciona una ficha con las teclas de función del teclado  
* Selecciona el destino de la ficha con una tecla numerica
```

Nota. Las tonalidades de la paleta de colores y el tamaño de la ventana podrían cambiar según la configuración del IDE.

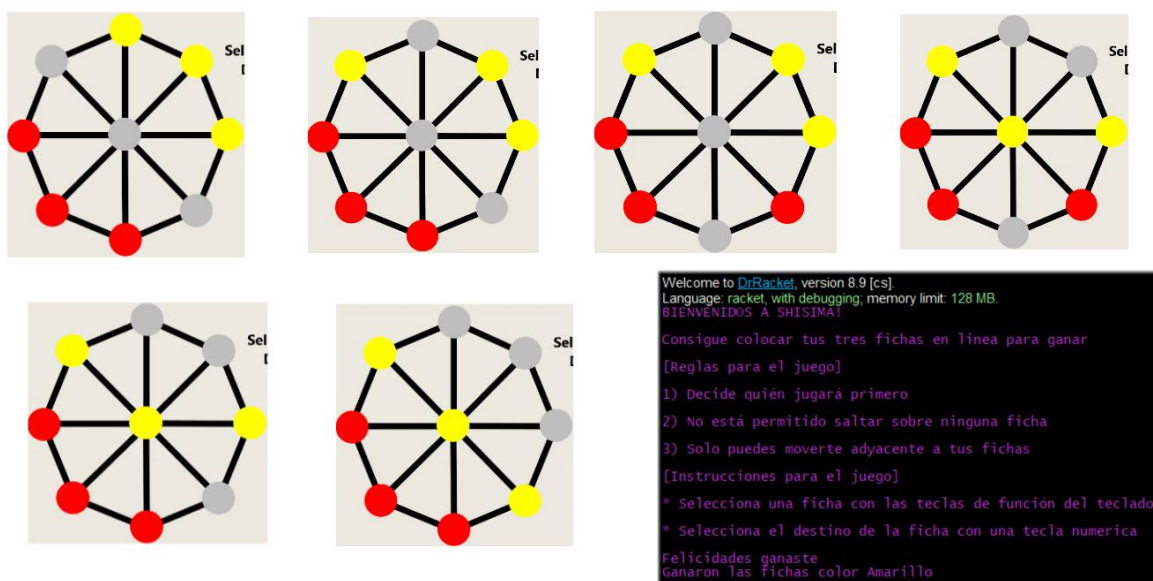
Cabe mencionar que, para la ejecución del juego, se debe correr el programa presionando el botón "Run" o usando el atajo de teclado "Ctrl + r". Una vez hecho esto, y teniendo en cuenta que la ruta de la imagen de fondo de la interfaz cambia según las bibliotecas del ordenador, el usuario podrá ejecutar el programa y jugar en un 1 vs 1 localmente.

Cada jugador será responsable de manejar tanto su tiempo de movimiento de ficha como de seleccionar correctamente el color de ficha correspondiente al seleccionado por él. Esto responde a emular las características esenciales de un juego de mesa donde los jugadores deben de corroborar que sus oponentes realicen sus operaciones bajo las reglas del juego y, en dado caso, no cometan trampa. Finalmente, el programa fue diseñado para generar mensajes de alerta cuando algún jugador este intentando mover una ficha a una posición no permitida. Por ejemplo, si un jugador intentase mover en una posición no adyacente a una de sus fichas, el programa desplegaría "movimiento no permitido" en la consola. A su vez, si el jugador intentase brincarse tanto una ficha suya como una de su oponente, el programa también desplegaría el mismo mensaje.

En la siguiente imagen se puede apreciar el desarrollo de una partida jugada con fines ilustrativos.

Imagen 5

Sucesión de partida de Shisima



Nota. Partida con fines ilustrativos

Referencias

Erica. (2022). Shisima: 3-in-a-Row Game from Kenya. What Do We Do All Day.
<https://www.whatdowedoallday.com/shisima/>