Instituto Tecnológico de Costa Rica

Área en Ingeniería en Computadores

CE-3104: Lenguajes, compiladores e intérpretes

Prof. Marco Rivera

Proyecto:

World Cup QaTec (WCQTec)

José Ignacio Calderón Díaz 2019031750
Abner Arroyo Quesada 2018103035
Max Garro Mora 2020024210

19 de octubre, 2022

1.1. Descripción de las funciones implementadas.

Las funciones utilizadas en la interfaz gráfica se centraron en mostrar los objetos en pantalla, definir sus características, entre otros. Se implementaron clases en la interfaz gráfica, ya que se tenía esa libertad (exclusivamente para la interfaz gráfica). Por lo que estas son las funciones:

1) (canvas frame)

Esta función se encarga de cargar las imágenes, que se encuentran en la carpeta "Resources", y las muestra en pantalla. En el caso de los jugadores, estos vienen con sus posiciones "x" y "y" definidas por el algoritmo genético, el área de juego es colocada en la posición equivalente a la esquina superior izquierda y el balón es colocado en la mitad del canvas, que es la mitad exacta de la cancha.

2) (posiniciales listajugs listacoord canvas):

Sus argumentos son listajugs (lista de jugadores) y listacoord (lista de coordenadas) y canvas (área donde se muestran los objetos)

Esta función es la encargada de crear las posiciones iniciales de todos los jugadores. Ya que los jugadores están dentro de una lista, esta va leyendo el primer elemento de la lista, una vez leído, emplea un cdr a listajugs y listacoords, y aplica un car a listajugs y listacoords.

3) (newpos listajugs listacoord):

Sus argumentos son listajugs (lista de jugadores) y listacoord (lista de coordenadas).

Esta función se encarga de realizar las animaciones de todos los jugadores. Primero revisa si las listas no son vacías y luego aplica la función de animación "moveObj" al primer elemento de la listajugs, al primer elemento del primer elemento de listacoord y al segundo elemento de la listacoord. Para luego recursivamente llamar los otros elementos, menos el primero, de listajugs y listacoord.

4) (moveObj player xNewPos yNewPos):

Sus argumentos son player (jugador), xNewPos (la nueva posición a la cual se dirige el jugador en x) y yNewPos (la nueva posición a la cual se dirige el jugador en y).

En este método se llama al método collisionBL, el cual se encarga de revisar las colisiones entre la bola y los límites de la cancha. En el método moveObj, se tiene el propósito que los jugadores se muevan solo en ciertos sectores de la cancha y que de ahí no se pasen. Esto lo hace con 8 condicionales, que simulan posibles movimientos que pueden tener, pero cada uno de estos

movimientos provoca que el canvas se refresque, para al final llamar de nuevo la función recursivamente y que sigue ejerciendo movimiento entre los jugadores.
Para el algoritmo genético, se utilizaron las funciones esperadas para este tipo de algoritmo: una función que da una población inicial con valores aleatorios, una función que selecciona a los ejemplares más aptos, una que hace una reproducción entre los individuos seleccionados y otra que agrega individuos con cambios aleatorios, para mantener la "diversidad genética". Todo ello se implementa exclusivamente de acuerdo al paradigma de programación funcional, donde las funciones son el enfoque y lo único que se declara son identificadores para las funciones y sus argumentos. Todos los ciclos se implementan de manera recursiva, utilizando tanto recursividad de pila como de cola, según fuera más conveniente. Las funciones principales son:

1) **poblacion_inicial**: esta función toma como entrada la alineación del equipo otorgada por el usuario al inicial la simulación. Esta retorna una lista con la alineación, una lista con tres veces la cantidad de defensas solicitados, una lista con tres veces la cantidad de mediocampistas seleccionados, una lista con tres veces las cantidad de mediocampistas seleccionados. Esta hace uso de la función **jugadores iniciales**, la cual recibe la cantidad de jugadores que se necesitan, la posición en la que juegan y una lista vacía en la cual retornará la cantidad de jugadores solicitados. Esta función necesita saber la posición del jugador, ya que usa esa información para poder seleccionar coordenadas aleatorias, pero dentro de los límites en los cuales dichos jugadores tienen permitido estar.
2) **fitness:** esta es la función de aptitud del algoritmo genético. Dada una población de individuos, esta los ordena de acuerdo a la aptitud de interés para cada posición (defensa, mediocampista o delantero) usando una función que implementa un quicksort descendiente, modificado para tomar en cuenta todas las aptitudes del jugador, pero principalmente la de interés para su posición:
   delanteros: ((interceptar [0-10] + velocidad [0-10] + punteria [0-10])/ 8) + punteria
   medios: ((interceptar [0-10] + velocidad [0-10] + punteria [0-10]) / 8) + velocidad
   defensas: ((interceptar [0-10] + velocidad [0-10] + punteria [0-10]) /) + interceptar

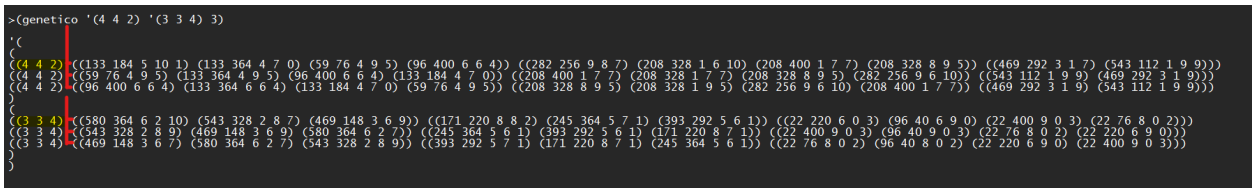
3) **crossover:** esta función se encarga de generar individuos a partir de la mezcla de individuos existentes en la población. Esta toma una población y cruza a los jugadores de una misma posición a partir de un punto de corte. Este punto de corte está dictado por el

tipo de jugador.. En los defensas, el punto de corte está en el valor de interceptar, en el de medios, velocidad y en el de delanteros en puntería.

1.2. Descripción de las estructuras de datos desarrolladas. Explicar cómo están compuestas sus listas.

Las estructuras de datos desarrolladas en el proyecto son las listas y las matrices. En Racket el uso de listas es fundamental. Es por ello que el uso de ellas está presente en varias partes del código.

Las primeras listas son las alineaciones de los equipos, estas se utilizan para identificar cuántos jugadores van en defensa, medio o delantero. Estas ingresan al algoritmo genético, donde se creará la población la cual se representa mediante una lista con sublistas que contienen las coordenadas y atributos de los jugadores. Finalmente el algoritmo genético retorna una lista con cada una de las generaciones de jugadores. En la siguiente imagen se puede observar un ejemplo para la siguiente entrada `(genetico '(4 4 2) '(3 3 4) 3)`



Todas las listas utilizadas manejan números enteros para realizar los cálculos respectivos en el algoritmo genético. Se observa una lista compuesta de los listados que contienen la información para cada equipo. En la imagen se observa, subayado en color amarillo, la alineación del equipo seguido por la lista que contiene a los jugadores. Estos se identifican con las flechas de color rojo.

Otra estructura de datos que se utiliza es la matriz, esta se utiliza en la parte gráfica del programa. Esto se debe a que para realizar los movimientos se necesita imaginar una matriz de 11 x 16 en el canvas. Es decir si necesitamos mover un objeto solo es necesario cambiarle la fila y columna de sus coordenadas.

1.3. Descripción detallada de los algoritmos desarrollados.

Algoritmo genetico

El algoritmo genético genera una población inicial aleatoria de jugadores. A esta primera población se le aplica una función de aptitud, la cual es un poco diferente para cada posición de jugador, ya que cada uno debe tener coordenadas y habilidades preferentes. Esta es la generación 1. Una vez se seleccionan los mejores candidatos estos pasan por un proceso de combinación y mutación, generando una nueva población, a la cual se le aplica de nuevo la función de aptitud, dando lugar a una nueva generación. Este proceso se repite hasta obtener la cantidad de generaciones deseadas.

Clase jugador y bola

Se desarrolla una clase jugador, la cual se encarga de almacenar las características de posición y composición. Dentro de sus atributos se encuentra la posición en el eje x, eje y, su nivel de puntería, su nivel de velocidad y su nivel de bloqueo. Para poder obtener y cambiar su información se realizan los métodos setters y getters.

Animación

La animación está representada mediante la función newpos, la cual recibe como parámetros la lista con los jugadores y la lista con las nuevas coordenadas en la matriz. Se realiza un chequeo que las listas no sean nulas y se procede a llamar la función de moveObj, esta se encarga de realizar la animación de cada jugador.

MoveObj toma como parámetros al jugador, la nueva coordenada en x y la nueva coordenada en y. Aquí se realizan las validaciones comparando la posición actual del objeto con la nueva posición, en caso que sea menor, esta debe sumar una unidad en la fila o columna. En caso que sea mayor esta se debe restar una unidad. Se obtienen 4 posibilidades, X Y mayores, X Y menores, X mayor Y menor, X menor Y mayor. Además, se tienen 2 posibilidades extra que es cuando una coordenada llega antes que la otra. La condición de Stop es la comparación de igualdad entre la posición actual del objeto con la nueva posición. Esta función se llama recursivamente hasta cumplir el movimiento. Finalmente, se actualiza el canvas para que las imágenes muestran las nuevas coordenadas.

1.4. Problemas conocidos: En esta sección se detalla cualquier problema que no se ha podido solucionar en el trabajo.

- Colisión de la bola: Se desarrolló un método que evaluará las colisiones de la bola con los límites y marco. Este si logra identificar cuando esta colisión fue realizada. Sin embargo esta presenta bugs a la hora de realizar el rebote, como parpadeo de la imagen y bucles en los movimientos. Además, la animación de la bola presenta un desborde de memoria por la cantidad de actualizaciones que se debe hacer, no se logro corregir la animación a tiempo.

1.5. Actividades planeadas por estudiante: Realizar un listado de las actividades que se deben realizar para completar el proyecto, estimar cuánto se va a tardar en esa actividad, quién la va a realizar y cuando la va a entregar.

Actividad	**Tiempo estimado**	**Encargado**	**Fecha de entrega**
Investigar: The Racket Graphical Interface Toolkit	1 dia	Ignacio Calderón	06/10/2022
Investigar: Algoritmos genéticos	1 dia	Abner Arroyo	06/10/2022
Creación de la ventana y mostrar objetos	1 dia	Ignacio Calderon	10/10/2022
Mostrar los jugadores	1 dia	Max Garro	10/10/2022
Animación jugadores	2 dias	Ignacio Calderón	13/10/2022
Animación bola	2 dias	Max Garro	13/10/2022
Poblacion inicial	1 dia	Abner Arroyo	15/10/2022
Función de aptitud	1 dia	Abner Arroyo	15/10/2022

Reproduccion	1 dia	Abner Arroyo	16/10/2022
Mutacion	1 día	Abner Arroyo	16/10/2022
Chequeo de colisiones con límites	1 dia	Max Garro	13/10/2022
Chequeo de colisiones con jugadores	1 dia	Max Garro	13/10/2022
Chequeo de colisiones con marco	1 dia	Max Garro	13/10/2022
Manual de usuario	2 dias	Ignacio Calderón Max Garro Abner Arroyo	18/10/2022
Documentación externa	1 dia	Ignacio Calderón Max Garro Abner Arroyo	18/10/2022

1.6. Problemas encontrados: descripción detallada, intentos de solución sin éxito, soluciones encontradas con su descripción detallada, recomendaciones, conclusiones y bibliografía consultada para este problema específico.

- **Problema en rendimiento de animación**

  Las animaciones se realizan actualizando cada fotograma lo cual si se realiza un movimiento de pixel por pixel este iba a generar un desbordamiento en memoria ya que la cantidad de llamadas recursivas iba a ser muy alta. Se realiza el proceso de debugging y se identifica que la función que estaba generando el problema en rendimiento es `(send canvas refresh-now)`. Este se encarga de actualizar el canvas, pero al ejecutarse tantas

veces el programa llegaba a comportarse lento. El primer intento fue colocar esta función de animación dentro de un hilo, por ello se procede a investigar el funcionamiento de los hilos en Racket. Se realizó la implementación del código con esta nueva característica , sin embargo, aún continuaba el problema de rendimiento.

Seguidamente, se decide reducir la cantidad de llamadas del método refresh-now, es por ello que se intenta colocar una vez al final del método de animación. Sin embargo, esto ocasiona una animación muy rápida donde nada mas se veia la posición inicial y la final. Por lo tanto, la otra manera de disminuir la cantidad de llamadas era moviendo los objetos más rápido. Siguiendo esta lógica se necesitaba realizar los movimientos como si fuera una matriz. Se realizan métodos de conversión entre la matriz y las coordenadas. Finalmente, se puso a prueba y el algoritmo de animación tenía un rendimiento mejor que con la anterior implementación.

- **Mover un objeto**

  Durante la lluvia de ideas para realizar la animación, se tuvo un problema al mover los objetos. El primer intento se quiso redibujar el objeto con las nuevas coordenadas hasta llegar a la posición final. Pero esto resultó en una cadena de objetos dibujados uno tras otro. El otro intento se decidió modificar solo las coordenadas de este, para ello se declaró una variable la cual iba a ser modificada. Se realiza el código pero durante pruebas el objeto no se movió, pero en consola se podía ver que las coordenadas si fueron variando. Después de un proceso de debugging se identificó que las variables no se pueden pasar como argumentos de una función, ya que Racket solo crea copias de las mismas cuando se pasa como parámetro. Así que para solucionar este problema se decidió crear una clase llamada jugador, la cual tiene como atributos sus coordenadas. Por lo tanto, cuando se dibuja una imagen, se le asigna las coordenadas directamente de la clase mediante un método get. Esto permite que cuando se refresque el canvas la imagen tenga nuevas coordenadas.

- **Colisiones del balón con los límites de la cancha y jugadores.**

Durante el trabajo se usó un método para mover a los jugadores. En cierto punto se consideró usar dicho método para también animar el movimiento del balón y que revisara sus colisiones, por lo que dependiendo de cuál fuera la colisión, el balón tendría un movimiento en específico. Lo que sucedió es que a la hora de emplear dicho método, el programa “crasheó”, ya

que consumía toda la memoria disponible, debido a que todo el código es recursivo, por lo que el número de llamadas a dicha función era excesivamente alto. Otra cosa que no consideramos fue que usar imágenes en vez de formas rectangulares iba a afectar el rendimiento del programa.

1.7. Conclusiones y Recomendaciones del proyecto.

Programar de una manera 100% funcional dificulta la legibilidad del código, si lo comparamos con un lenguaje como Java o Python. Al no declararse variables hay que basarse en comentarios, nombres de funciones y comprensión profunda del código. Además, la longitud del código puede llegar a ser mayor dependiendo de la naturaleza del problema. En el caso del algoritmo genético, una implementación sencilla en Python puede realizarse en alrededor de 100 líneas de código. Sin embargo, la implementación realizada en este proyecto tuvo una longitud de alrededor de 500 líneas de código.

La interfaz de Racket es poco flexible, además que muy rudimentaria comparada con las interfaces de otros lenguajes. El hecho que todos los métodos hayan sido programados recursivamente excedía la capacidad de memoria de DrRacket, por lo que crear el programa fue una tarea difícil, ya que hubo que pensar cómo optimizar el uso de memoria, lo cual en otros lenguajes e IDEs no era del todo necesario.

Cargar imágenes en vez de usar formas rectangulares para representar a los jugadores y el balón provocó que se consumiera mucha memoria vital para otros métodos de animación, los cuales eran necesarios para lograr completar el programa solicitado.

1.8. Bibliografía consultada en todo el proyecto

Racket. (2017, 08 15). racket-lang.org. Retrieved from Racket: https://racket-lang.org/

Racket. (2017, 08 15). The Racket Graphical Interface Toolkit. Retrieved from Racket: http://download.racket-lang.org/releases/6.9/doc/gui/

GeeksforGeeks. (s.f).Genetic Algorithms.Retrieved from: https://www.geeksforgeeks.org/genetic-algorithms/

Arroyo. Abner, Brown Nasser.(2022).TicTacToe. Retrieved from https://github.com/N4SSER/TicTacToe/blob/main/GUI.rkt

Racket.(2018, 08, 15).Generic Numberics. Retrieved from https://docs.racket-lang.org/reference/generic-numbers.html

Soegaard.(2015, 03, 10). Breakout. Retrieved from https://github.com/soegaard/breakout

Anexo

Link al repositorio

https://github.com/Abner2111/WCQaTEC.git

Bitacora

(05/10/2022)

Todos: Creamos el repositorio, grupo en discord y whatsapp para mejorar la comunicación entre los 3.

Max: me dediqué a investigar sobre algoritmos genéticos y también a buscar repositorios en github, que lo empleaban.

Ignacio: hicimos una pequeña lluvia de ideas sobre cómo implementar el algoritmo genético.

Abner: Aporté feedback sobre trabajos anteriores en racket, creo el repositorio para desarrollar el proyecto y añado a Ignacio y a Max al mismo. Decidimos la manera en que se va a implementar el algoritmo genético, teniendo una función de aptitud que valida cada tipo de jugador por separado.

(06/10/2022)

Max: seguí investigando sobre algoritmos genéticos, pero en menor medida, ya que tengo trabajos de otros cursos. Además que conversé con varios amigos, que ya han pasado este curso y me mostraron sus repositorios para tener una idea de cómo hacer la interfaz gráfica.

Ignacio: Vi videos sobre algoritmos genéticos y repasé las presentaciones de Algoritmos y estructuras de datos II.

Abner: Investigué por medio de videos, páginas web y literatura sobre algoritmos genéticos y profundicé en cada uno de los apartados.

(07/10/2022)

Max: revisé los repositorios de mis amigos y tengo una idea de cómo hacer la interfaz gráfica. Aunque usen otro tipo de algoritmos, estos repositorios son útiles para nuestro proyecto.

(08/10/2022)

Todos: Nos reunimos por primera vez en discord y discutimos sobre cómo nos vamos a repartir el trabajo y tener deadlines para terminar las tareas planteadas.

Max: Me toca hacer la interfaz gráfica con Ignacio, pero él va a empezar solo, ya que yo estoy muy lleno de trabajos, pero cuando termine, me incorporo para ayudarle.

Ignacio: Comencé la interfaz gráfica, basándome en la documentación de racket. Intenté buscar videos o explicaciones sobre GUI en Racket pero no encontré mucho. Además, añadí el .gitignore

Abner:

(09/10/2022)

Max: no pude hacer nada relacionado al proyecto porque tenía que terminar mis tareas.

(10/10/2022)

Grupos: Nos reunimos en discord para discutir sobre cómo va a funcionar el algoritmo genético y también plantear las tareas específicas de la interfaz gráfica, para así dividirlas entre Max e Ignacio.

Max: Me toca hacer el método que ubica a los jugadores en sus posiciones iniciales. Eso lo haré entre hoy y mañana, cuando termine esto, haré otra tarea disponible.

Ignacio: Se crea la ventana con la cancha y marcadores, queda listo la actualización del marcador

(11/10/2022)

Max: Me reuní con Ignacio para observar los repositorios de mis amigos, y también me ayudó para terminar el método que estaba haciendo desde ayer.

Ignacio: Realice la clase Jugador y el primer intento del algoritmo de animación. Se realizan pruebas con imágenes y rectángulos.

(12/10/2022)

Max: Me reuní con Ignacio para ver cómo se coloca un equipo entero en una lista, para así hacer las pruebas necesarias y verlas representadas en la interfaz gráfica.

(13/10/2022)

Max: Me reuní con Ignacio para trabajar en la interfaz gráfica, y logré terminar el método para colocar a los jugadores en sus posiciones iniciales.

Ignacio: realice todos los sprites de los jugadores en Paint. Se trabajó en la GUI con Max. Se añade al método de animación todos los posibles cambios de coordenadas comparando los valores nuevos con los viejos.

Abner: Definimos el formato de salida para el algoritmo genético y así poder implementarlo en la interfaz.

(14/10/2022)

Max: Me reuní con Ignacio para trabajar en la interfaz gráfica y resolver problemas que se tenían con ciertos métodos con errores.

(15/10/2022)

Max: hablé con Ignacio por whatsapp y quedamos en ir adelantando la documentación técnica, y terminar lo antes posible la interfaz gráfica para ayudarle a Abner con el algoritmos genético.

Abner: Subo mis avances en el desarrollo del algoritmo genético a github. Creo una función que crea una población a partir de la alineación del equipo dada por el usuario. Esta agrega valores aleatorios, pero dentro de las posibilidades de cada posición de jugador. Agrego una función de aptitud llamada fitness. Esta se apoya en un algoritmo quicksort para ordenar de más apto a menos apto los jugadores de cada posición y selecciona la cantidad dictada por la alineación del equipo.

(16/10/2022)

Max: hice la clase ball, que contiene los getters y setters de la bola, además que la coloqué en la interfaz y se muestra bien. Ahora falta verificar las colisiones con los marcos, límites de la cancha y jugadores.

(17/10/2022)

Max: me reuní con Ignacio un rato, ya que llegué muy tarde a mi casa debido a un quiz de tutorías de un curso y solo vimos los límites de la cancha y ya.

Abner: Investigar el funcionamiento de la función de crossover. Encuentro una explicación sencilla y gráfica en geeksforgeeks.org. Agrego la función de crossover/reproducción para poder producir descendencia a partir de una generación.

(18/10/2022)

Todos: nos reunimos en discord para explicarnos el código y entender qué hace para no repetir código, además de terminar ciertos apartados que nos faltan.

Max: hoy trabajé las colisiones de la bola con los límites de la cancha, los marcos y los jugadores.

Abner: Trabajo en implementar la función de mutación, sin embargo no tengo éxito debido a la longitud de esta y la difícil legibilidad del código. Arreglar bugs en la función de crossover, ya que esta estaba cruzando individuos de distintas posiciones de juego, lo cual provocaba resultados erróneos. Actualizo los límites de posiciones para los jugadores en la función de población inicial, ya que por un error de comunicación entre Ignacio y yo los valores eran completamente erróneos.

(19/10/2022)

Abner: Implementar la función de mutación. Después de dormir, pensar bien el algoritmo a seguir y escribir un pseudocódigo. Además se crea la función que realiza el ciclo del algoritmo genético hasta completar las generaciones requeridas.

Max: Trabajé en la documentación técnica y haciendo el manual de usuario. En la madrugada intenté hacer que la bola se moviera cuando colisionaba con los jugadores y límites de la cancha, pero no pude por la limitante de memoria.