INSTITUTO TECNOLÓGICO DE COSTA RICA

Ingeniería en Computadores

Algoritmos y Estructuras de Datos II

Proyecto #2

“GeneticGladiador”

Roy Acevedo Méndez

Carné: 2013095023

Marycruz Arlyn Céspedes Lobo

Carné: 2016136489

Pablo Esquivel Morales

Carné: 2014016438

Steven Vega Zúñiga

Carné: 2014068428

2017

Contenido

[INTRODUCCIÓN 2](#_Toc480736282)

[DESCRIPCIÓN DEL PROBLEMA 3](#_Toc480736283)

[PLANIFICACIÓN Y ADMINISTRACIÓN DEL PROYECTO 3](#_Toc480736284)

[Historias de Usuario 3](#_Toc480736285)

[Distribución de Historias de Usuario por Criticalidad y Secuencia de Uso 4](#_Toc480736286)

[Minimal System Span 4](#_Toc480736287)

[Plan de Iteraciones 4](#_Toc480736288)

[Asignación de User Stories 5](#_Toc480736289)

[Descomposición de User Stories en Tareas 5](#_Toc480736290)

[Bitácora 5](#_Toc480736291)

[DISEÑO 5](#_Toc480736292)

[Diagrama de Arquitectura 5](#_Toc480736293)

[Diagrama de Componentes 5](#_Toc480736294)

[Diagrama de Despliegue 5](#_Toc480736295)

[Diagrama de Clases 5](#_Toc480736296)

[Diagrama de Secuencia 6](#_Toc480736297)

[Definir fitness de cada individuo 6](#_Toc480736298)

[Definir el mejor gladiador de cada generación 6](#_Toc480736299)

[Actualizar la Resistencia Total del Gladiador luego de recibir un golpe 6](#_Toc480736300)

[IMPLEMENTACIÓN 6](#_Toc480736301)

[Descripción de Bibliotecas Utilizadas 6](#_Toc480736302)

[Descripción de las Estructuras de Datos Desarrolladas 6](#_Toc480736303)

[Descripción Detallada de los Algoritmos Desarrollados 7](#_Toc480736304)

[Problemas Encontrados 7](#_Toc480736305)

[CONCLUSIÓN 7](#_Toc480736306)

[BIBLIOGRAFÍA 7](#_Toc480736307)

# 

# INTRODUCCIÓN

En el siguiente proyecto se planteará una solución programada en C++ orientada a objetos para realizar un videojuego estilo *fighter* ambientado en la Antigua Roma con el objetivo de implementar y diseñar dicha solución mediante la utilización de algoritmos genéticos y *backtracking*, utilizar diagramas de clases UML para modelar una solución al problema e implementar una arquitectura Cliente-Servidor; por último, fomentar la creatividad del estudiante mediante el análisis y diseño de algoritmos.

# DESCRIPCIÓN DEL PROBLEMA

El problema consiste en implementar un juego estilo *Fighter* ambientado en la Antigua Roma en C++ para desktop. El juego debe gestionar dos poblaciones de gladiadores, de las cuales, en cada partida (generación), debe tomar el individuo con mejor fitness de cada una y hacerlos atravesar una Zona de Intimidación. Para finalmente enfrentarse entre ellos en el Coliseo Romano.

# PLANIFICACIÓN Y ADMINISTRACIÓN DEL PROYECTO

## Historias de Usuario

**La Zona de Intimidación**

* Como usuario quiero que el juego tenga un lugar por donde pasen los gladiadores antes de pelear en el coliseo, que mida 10x10, y que tenga un único punto de ingreso y otro de salida.

**Los Obstáculos**

* Como usuario quiero obstáculos en la zona de intimidación, y que esos obstáculos sean torres que lancen flechas que disminuyan la resistencia de los Gladiadores cuando atraviesen esa zona.
* Además quiero que por cada partida se coloquen 3 torres que disparen 1 posición de distancia en cualquier dirección.

**Los Gladiadores**

* Quiero poder contar con una población de gladiadores pero utilizar el mejor.
* Quiero que se generen más gladiadores a partir de los mejores.
* Además como usuario quiero que los gladiadores encuentren la ruta para llegar al coliseo por la zona de intimidación.

**El Combate entre Gladiadores**

* Quiero que los gladiadores puedan combatir una vez que lleguen al coliseo y poder escoger el tipo de golpe a utilizar.
* Quiero que ese golpe cause impacto en la resistencia del gladiador, dependiendo del tipo de golpe.
* Quiero que el gladiador pierda al quedarse sin resistencia ya sea en la zona de intimidación o en combate.
* Gana el último en quedar en pie, si los dos mueren ninguno gana.

**Visualizador de estadísticas**

* Como usuario quiero poder ver estadísticas como generaciones transcurridas, victorias en el combate final, fitness del jugador seleccionado, fitness promedio de la generación, probabilidad de mutaciones y cantidad de mutaciones ocurridas.

**El Evolucionador**

* Quiero que el evolucionador esté escrito en Java y sea un RESTful API y que controle el algoritmo genético.
* Quiero que guarde la información de las generaciones en un XML y así consultar en cualquier momento.

**XML**

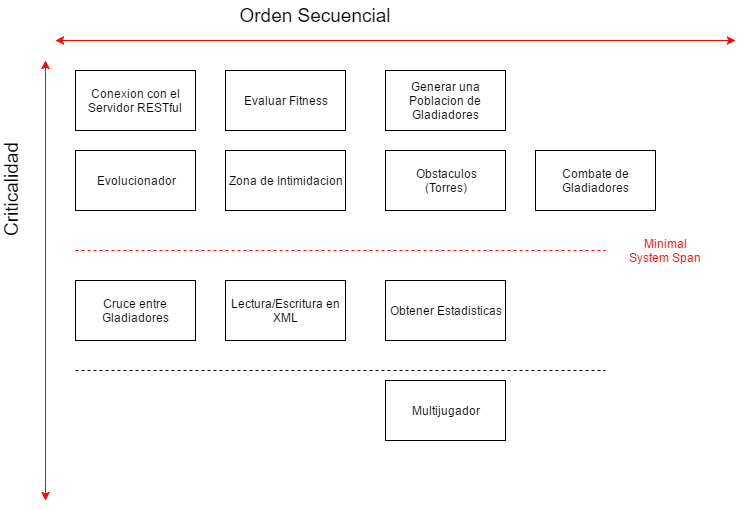
Por cada población de gladiadores quiero que los XML contengan los siguientes elementos:

* Información genética de cada individuo de la generación.
* Número de generación.
* Cantidad de mutaciones e inversiones ocurridas.
* Gladiador seleccionado por generación.
* Resultado de los combates finales.

## Distribución de Historias de Usuario por Criticalidad y Secuencia de Uso

## Minimal System Span

Luego de analizar cada historia de usuario, se concluyó que el minimal system span debía ser conformado por: la conexión RESTful, el evolucionador, evaluar fitness, generar población de gladiadores, la zona de intimidación, los obstáculos y el combate entre gladiadores.



## Plan de Iteraciones

Historia de Usuario Tiempo

## Asignación de User Stories

## Descomposición de User Stories en Tareas

## Bitácora

|  |  |
| --- | --- |
| Lunes 17 de abril-Domingo 23 de abril | * Se conversó sobre el proyecto. * Se trabajó en la documentación del proyecto. * Se comenzó con la creación de la clase Gladiador. |
| Lunes 24 de abril- Domingo 30 de abril | * Se trabajó en el Algoritmo genético. * Se trabajó en interfaz. * Se trabajó en el servidor. |
| Lunes 1 de mayo – Lunes 7 de mayo | * Se trabajó en interfaz. * Se trabajó en el servidor. * Se trabaja en el pathfinding. |
| Lunes 8 de mayo – Viernes 12 de mayo | * Se trabaja interfaz. * Se trabaja en el cliente. * Se trabaja en documentación. * Se trabaja en el servidor |

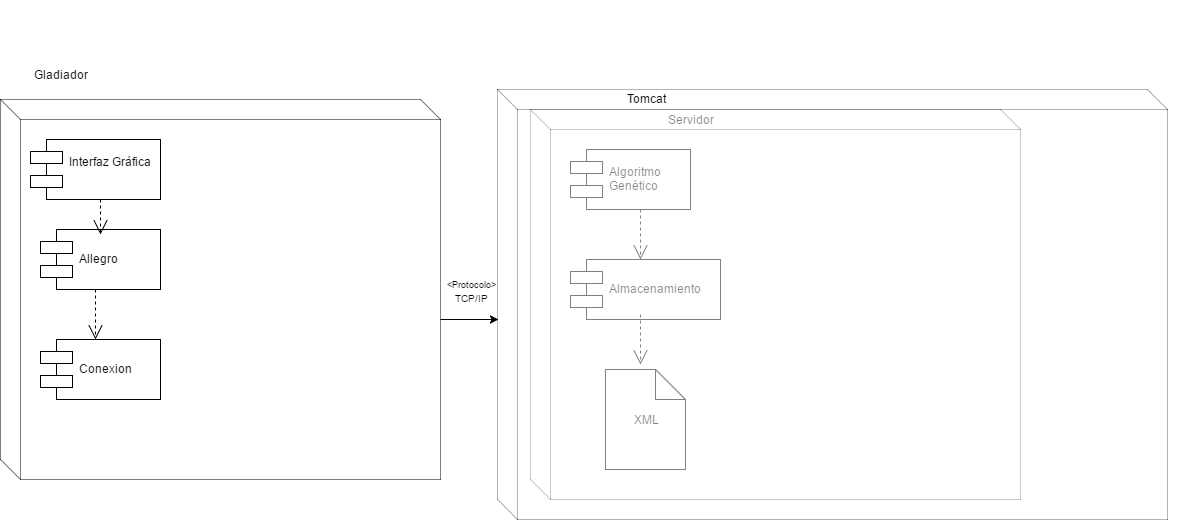
# DISEÑO

## Diagrama de Arquitectura

El usuario interactúa con aplicación por medio de la GUI, la lógica del programa contiene el algoritmo de pathfinding y muestra a los gladiadores cuando pasan por la zona de intimidación y luego de eso el enfrentamiento entre los gladiadores. La aplicación obtiene la información de los gladiadores seleccionados por medio del servidor, al que se conecta por medio de acceso a internet. El servidor contiene la lógica del algoritmo genético y se encarga de escribir la información de la población en archivos XML.

## Software Architecture GG (1).png

## Diagrama de Componentes

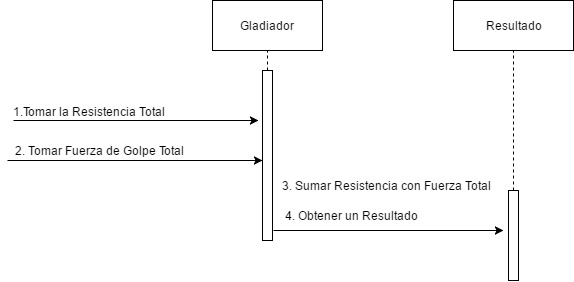


## Diagrama de Despliegue

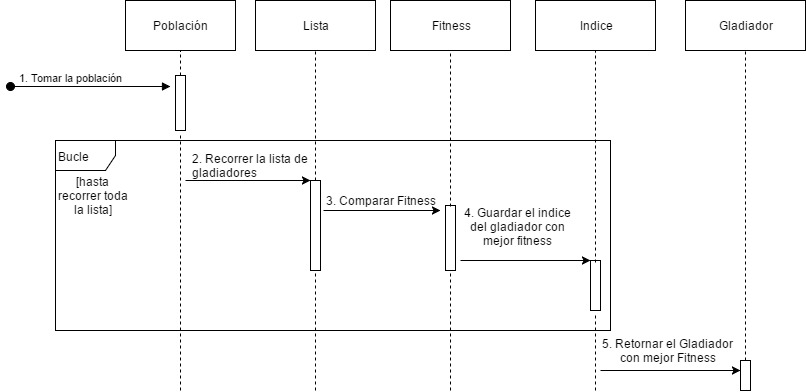
## Diagrama de Clases

## Diagrama de Secuencia

### Definir fitness de cada individuo



### Definir el mejor gladiador de cada generación



### Actualizar la Resistencia Total del Gladiador luego de recibir un golpe

# IMPLEMENTACIÓN

## Descripción de Bibliotecas Utilizadas

***ALLEGRO:*** es una biblioteca libre y de código abierto para la programación de videojuegos en el lenguaje C; pero bibliotecas adicionales permiten su implementación en otros lenguajes como C++, Java, C#, Visual Basic.NET, Python, D, Lua, Pascal, Perl, Ruby, Go, Scheme, Common Lisp, Ocaml, Forth y Mercury. Esta cuenta con funciones para manipular texto, imágenes, gráficos, sonidos, dispositivos de entrada, y además acceso al sistema de archivos.

***JAX-RS:*** es una API del lenguaje de programación Java que proporciona soporte en la creación de servicios web de acuerdo con el ​​estilo arquitectónico Representational State Transfer (REST).1 JAX-RS usa anotaciones, introducidas en Java SE 5, para simplificar el desarrollo y despliegue de los clientes y puntos finales de los servicios web. Esto se implementó en el desarrollo de los XML.

***Apache Tomcat:*** Es un contenedor de Java de código abierto desarrollado por Apache Software Foundation (ASF). Tomcat implementa varias especificaciones de Java EE, incluyendo Java Servlet, JavaServer Pages (JSP), Java EL y WebSocket, y proporciona un entorno de servidor HTTPweb "puro Java" en el que se puede ejecutar código Java.

## Descripción de las Estructuras de Datos Desarrolladas

***Listas:*** es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior.

## Descripción Detallada de los Algoritmos Desarrollados

## Problemas Encontrados

No se logra la comunicación cliente-servidor.

Se tienen problemas al implementar el pathfanding.

Se presentaron errores al instalar bibliotecas.

Falta de tiempo para realizar y completar todas las historias de usuario.

# CONCLUSIÓN

* Se investigó sobre el algoritmo de Pathfinding A\*.
* Se investigó sobre los Servicios Web REST.
* Se investigó sobre algoritmos genéticos.
* Se desarrolló un algoritmo genético que evaluaba las características de una población aleatoria y genera nuevos individuos a partir de los mejores de la población.
* Se desarrolló un servidor RESTful para efectuar la conexión con el cliente y transmitir la información de los individuos seleccionados.
* Se creó una aplicación que pone a prueba los individuos seleccionados por el algoritmo genético.
* Se implementó un lector y escritor de archivos XML para almacenar los datos de la población creada por la aplicación.

# BIBLIOGRAFÍA

Amatellanes.wordpress.com. (2017). *GitHub >> Instalación y primeros pasos en GitHub usando Ubuntu*. Disponible en: https://amatellanes.wordpress.com/2013/03/10/107/

Crysol.org. (2017). *HTTP GET con libcurl en C++ | CRySoL*. Disponible ent: http://crysol.org/es/node/800

Curl.haxx.se. (2017). *libcurl - source code examples*. Disponible en: https://curl.haxx.se/libcurl/c/example.html

Curlpp.org. (2017). *curlpp by jpbarrette*. Disponible en: http://www.curlpp.org/

Drupal.org. (2017). *Calling a REST server with Command Line cURL (login, create comment, logout)*. Disponible en: <https://www.drupal.org/node/1795770>

Ubuntuforums.org. (2017). *[SOLVED] Codeblocks c++ linking curl lib ?*. Disponible en: https://ubuntuforums.org/showthread.php?t=1743471