Aplicación Software para el Desarrollo de Prácticas de Física Mecánica

Trabajo de Grado

Mónica Andrea Camacho Dorado
Wilson Harvey Imbachi Meneses

Director: Ing. Francisco Franco Obando Díaz
Codirector(a): Ing. Judy Cristina Realpe Chamorro

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Electrónica, Instrumentación y Control.
Popayán, octubre de 2016

Contenido

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

INTRODUCCIÓN

Hoy en día el continuo desarrollo de nuevas y variadas tecnologías ha originado un cambio sustancial en el pensamiento general de las personas, convirtiendo en una necesidad la realización de procesos de formación continua como base para el mantenimiento de lo que hoy se conoce como sociedad de la información [1]. Gracias a algunas herramientas de difusión como internet, la información se encuentra al alcance de todo el mundo, generando mayor o menor impacto dentro de múltiples contextos, entre ellos la educación.

La incorporación de las tecnologías de la información y la comunicación (TIC) al ámbito de la educación ha ido tomando fuerza con el pasar de los años, cumpliendo una función importante como facilitadoras de la gestión pedagógica e impulsando la innovación y transformación de los ambientes educativos gracias a la integración de una gran variedad de herramientas dentro de los procesos de enseñanza y aprendizaje [2]. Dichas herramientas aportan algunas ventajas dirigidas a contribuir positivamente en el proceso educativo, ya que, entre otras cosas, permiten el desarrollo de habilidades a través de la ejercitación gracias a la simulación de procesos con diferentes grados de complejidad, facilitan el trabajo independiente de los estudiantes, enriquecen el campo de la pedagogía al incorporar elementos tecnológicos dentro de los métodos de enseñanza, entre otras [3].

En el caso de la Universidad del Cauca, concretamente dentro de los programas de ingeniería, se tiene a disposición una innumerable cantidad de recursos software para complementar y dar soporte a los contenidos teóricos de un gran número de asignaturas. Muchas de esas herramientas han sido adquiridas por la Universidad bajo el pago de una licencia de funcionamiento, otras en cambio han sido desarrolladas por docentes y estudiantes de las mismas facultades como parte de algunos trabajos de investigación.

Aunque dentro de estos programas se tiene a disposición un buen número de herramientas de simulación, no todas las asignaturas cuentan con un paquete software o un simulador para la realización de prácticas virtuales. Tal es el caso del curso de física mecánica dictado dentro de los programas de ingeniería, el cual, en algunos casos, no cuenta con actividades de experimentación para llevar a cabo prácticas relacionadas con las temáticas impartidas en las clases. Es por ello que a través del siguiente trabajo de grado se busca la implementación de una herramienta software que permita a los estudiantes la experimentación, por medio de prácticas virtuales, de algunos de los conceptos más importantes tratados dentro del curso de física mecánica de la Universidad del Cauca.

El proceso llevado a cabo para el logro de este objetivo se muestra a través del presente documento, el cual está estructurado de la siguiente manera: En el capítulo 1 se realiza la selección de los elementos de desarrollo necesarios para la implementación de una herramienta que permita realizar prácticas de física mecánica por medio de simulaciones basadas en animaciones. En el capítulo 2 se aborda la descripción técnica del proceso de implementación de la herramienta, partiendo de la definición de los sistemas individuales que conforman cada una de las prácticas que serán incluidas en ella. En el tercer capítulo se presenta el producto final desarrollado a partir de las herramientas escogidas, destacando su correcta funcionalidad y veracidad en los datos proporcionados al usuario después de cada proceso de simulación. Por último dentro del capítulo 4 se presentan las conclusiones del proyecto.

Adicionalmente el trabajo cuenta con una serie de anexos que complementan los temas relacionados con la presente monografía, los cuales corresponden a: ecuaciones de movimiento de los sistemas físicos diseñados (Anexo A), guías de actividades para la realización de prácticas de física mecánica (Anexo B), guía de pasos para alojar la aplicación dentro del servidor web escogido (Anexo C), configuración de un cliente VPN (Anexo D), un manual del programador para las prácticas de física mecánica (Anexo E), formato de la encuesta de manejo de la aplicación dirigida a los estudiantes de los cursos de física mecánica (Anexo F).

Capítulo 1

HERRAMIENTAS DE SOPORTE TECNOLÓGICO

A continuación se analiza la importancia de incorporar las herramientas de simulación al ámbito educativo, en especial dentro del área de la física, y se exploran las opciones disponibles para llevar a cabo el desarrollo de una aplicación que permita la realización de simulaciones basadas en sistemas físicos representados por medio de animaciones gráficas en dos dimensiones.

1.1. HERRAMIENTAS DE SIMULACIÓN EN LA ENSEÑANZA DE LA FISICA.

Hoy en día las TIC cumplen un papel muy importante dentro del contexto de la educación, integrando a los procesos de enseñanza y aprendizaje una gran variedad de herramientas que brindan al estudiante la posibilidad de profundizar y complementar muchos de los temas relacionados con las áreas impartidas en su entorno educativo [4]. Dicha evolución se presenta como respuesta al panorama educativo actual, en el cual se plantean nuevos retos académicos que impulsan la inclusión de nuevas metodologías que permitan a los estudiantes una mayor autonomía en el proceso de aprendizaje [5].

Gracias a la masificación de la computadora, la utilización de este tipo de recursos ha adquirido una notable importancia desde el punto de vista pedagógico, ya que han servido como puente entre las actividades de enseñanza y asimilación cognitiva por parte de los estudiantes. Gracias a las aplicaciones informáticas orientadas a la educación se han generado nuevas formas personales de interacción y recepción del conocimiento, dando la posibilidad de llevar a cabo diferentes tareas y facilitando su uso pedagógico en múltiples campos [4].

Entre los recursos digitales implementados con fines educativos, las herramientas de simulación se destacan por su impacto visual y sus características de animación, pues permiten fundamentalmente la representación de un ambiente similar al de un laboratorio real. Este tipo de software, utilizado como un elemento adicional de refuerzo, brinda apoyo a los estudiantes en el proceso de consolidación del conocimiento de manera autónoma, además, como herramienta didáctica, ayuda a

fomentar un entorno participativo y constructivista entre alumnos y maestros dentro del aula de clase. De manera adicional, a través del uso de este tipo de recursos se potencializa la inclusión de competencias relacionadas con el manejo de las tecnologías de la información y la comunicación (TIC), tan importantes hoy en día para la formación de los estudiantes [5].

Por otra parte, el potencial dado por una herramienta de simulación permite robustecer y complementar las capacidades cognitivas que involucran los alumnos cuando se enfrentan a la solución de una tarea específica, fomentando en ellos la capacidad de visualizar, organizar y automatizar ciertos procesos de un nivel inferior, de modo que puedan concentrar su actividad cognitiva en la solución de los problemas centrales propuestos dentro de las diferentes actividades educativas en las que se interactúa con el medio de aprendizaje [6].

Las herramientas de simulación facilitan la representación dinámica del funcionamiento general de un sistema, permitiendo la visualización de sus procesos y de su evolución a través de la interacción con sus distintos componentes. Dicha interacción, traducida en manipulación y transformación de objetos dentro del espacio de la interfaz, brinda a las actividades educativas un fortalecimiento pedagógico, ya que involucra un conjunto distinto de competencias cognitivas comparado con la utilización de otros recursos [7].

Dentro del contexto educativo, las herramientas de simulación en general son aplicables en muchas de las áreas del conocimiento, siendo la física una de las ciencias en donde más se han aprovechado los beneficios y potencialidades que ofrecen las TIC en cuanto al desarrollo de nuevas metodologías encaminadas a la enseñanza y el aprendizaje. El trabajo con el software puesto al servicio de la física da como resultado un conjunto de elementos complementarios, experimentales y cercanos a los estudiantes, los cuales aportan las siguientes ventajas [8]:

- Permiten la representación idealizada, dinámica y visual de infinidad de fenómenos físicos y experimentos que pueden llegar a ser costosos, peligrosos o no factibles de realizar dentro de un laboratorio convencional, ya que por medio de un ordenador se puede mostrar una versión simplificada del mundo natural, en la que los estudiantes podrán concentrar toda su atención en los detalles importantes de cada simulación.

- También brindan la posibilidad, tanto a profesores como estudiantes, de visualizar objetos y procesos que están más allá de su control en el mundo real, con la ventaja de que pueden explorar situaciones hipotéticas e interactuar con

una versión simplificada de un proceso o sistema que acepta la variación de cualquier tipo de parámetro.

- Las simulaciones virtuales se basan en modelos matemáticos que representan con precisión los fenómenos y sistemas que son objeto de estudio, por lo tanto a partir de una simulación bien diseñada el estudiante se verá involucrado en una serie de procesos que lo llevarán a tratar de predecir el curso o el resultado de las acciones realizadas, buscando identificar los factores que determinan la ocurrencia de los eventos y motivando su lado crítico a partir del razonamiento adecuado de dichas acciones [9].

- En términos pedagógicos, existen grandes beneficios al trasladar al estudiante a un ambiente virtual, ya que, entre otras cosas, favorece la autonomía de su aprendizaje al darle la posibilidad de personalizar los experimentos físicos por medio de variables propias que arrojarán resultados que puede llegar a compartir con los demás miembros de su grupo, lo que resulta en una experiencia más enriquecedora si se compara con un laboratorio convencional, en el que se debe seguir de manera estricta un procedimiento rígido y poco flexible [5].

- Los recursos electrónicos complementan el componente experimental de los cursos de física, mejorando la eficiencia en la asimilación de las lecciones impartidas. El uso de este tipo de herramientas permite aislar lo más importante dentro del estudio de un fenómeno, separar factores secundarios, identificar algunos patrones, probar los experimentos una y otra vez con parámetros variables, guardar y volver a la investigación en un momento posterior, sin que hayan cambios en las condiciones de experimentación [10].

- La experimentación virtual se implementa por medio del modelo informático de una ley, un fenómeno o un proceso, lo que ofrece nuevas oportunidades educativas a los estudiantes, llevándolos a un estado de participación activa dentro de los procesos y dejando de lado su papel de simples observadores.

- Las demostraciones interactivas, apoyadas en la presentación correcta de los cálculos que soportan los sistemas implementados, favorecen por igual a estudiantes y docentes, ya que contribuyen a mejorar el estudio auto dirigido por parte de los estudiantes y representan una herramienta didáctica muy poderosa con la que los profesores pueden exponer los conceptos teóricos y prácticos de una determinada temática, todo en un mismo proceso de experimentación [11].

- Por medio de las herramientas de simulación se ha logrado minimizar, en mayor o menor medida, el carácter abstracto de algunas temáticas estudiadas dentro de los cursos de física, las cuales en ocasiones llegan a ser incomprensibles si son impartidas a través de los métodos convencionales. En este sentido tales herramientas otorgan la facilidad de transmitir esos mismos conceptos por medio de simples representaciones visuales e interacciones sin modificar la naturaleza propia de los mismos. Estas representaciones visuales dan a los estudiantes las herramientas necesarias que desarrollan su intuición, incluso en situaciones en las que se enfrentan a problemas y sistemas complejos que no tienen analogía alguna con su vida cotidiana [12].

Estas ventajas permiten apreciar las herramientas TIC como un recurso innovador, que al ser integradas de manera complementaria en la enseñanza de la física se convierten en un elemento pertinente y acertado que brinda a los estudiantes la posibilidad de conocer desde otra perspectiva los conceptos importantes contenidos dentro de cada temática, permitiendo actividades de interacción con sistemas físicos variados y obteniendo resultados con un alto grado de confiabilidad.

Un ejemplo de estas herramientas son los sitios web que incluyen *applets,* es decir, pequeños programas disponibles a través de internet con los que se modelan y simulan diferentes experimentos físicos (o de cualquier otra ciencia aplicada) partiendo de algunas condiciones similares a las que se tienen dentro de un laboratorio convencional. Gracias a estos elementos los estudiantes tienen la libertad de realizar infinidad de veces los procesos experimentales, variar parámetros y observar las respuestas del sistema ante dichos cambios con el fin de establecer una relación entre lo que se hace en la realidad y lo que muestra la herramienta de simulación [5]. En el siguiente apartado se describen algunas de las herramientas de simulación disponibles en el mercado actual, las cuales permiten la realización de prácticas virtuales de física mecánica.

1.2. PLATAFORMAS PARA LA REALIZACIÓN DE PRÁCTICAS VIRTUALES DE FÍSICA MECÁNICA

La exploración de las plataformas disponibles para la simulación de experimentos de física mecánica arrojó como resultado un listado general de herramientas con diversas características en cuanto a representación de los fenómenos y sistemas físicos, despliegue de resultados e interactividad con los usuarios, las cuales se muestran a continuación:

1.2.1. GNU FisicaLab

FisicaLab es una herramienta educativa de software libre creada a partir de distribuciones GNU/Linux para la solución de variados problemas físicos, la cual permite a los usuarios un enfoque completo hacia algunos de los conceptos más importantes de la física y brinda soporte a todo el componente matemático que incluyen los diferentes fenómenos estudiados a nivel de simulación. La última versión de FisicaLab incluye los siguientes módulos relacionados con la rama de la física mecánica: cinemática de partículas en dos dimensiones (2D), cinemática circular de partículas en 2D, estática de partículas en 2D, estática de cuerpos rígidos en 2D, dinámica de partículas en 2D y dinámica circular de partículas en 2D. Los problemas de estática y dinámica planteados en cada simulación parten de la utilización de diagramas de cuerpo libre en cada uno de los elementos involucrados en los diferentes sistemas físicos [13].

FisicaLab pone a disposición de sus usuarios la documentación necesaria con la descripción completa de cada uno de los elementos de la interfaz gráfica y una serie de ejemplos demostrativos, resueltos paso a paso, que facilitan la comprensión y el aprendizaje rápido de los fenómenos estudiados. También cuenta con una comunidad disponible para la solución de problemas relacionados con el uso e instalación de FisicaLab (FisicaLab-Discusión) y otra para discutir aspectos de desarrollo, petición de mejoras y reporte de errores de la herramienta (Bug-FisicaLab). El software está disponible para sistemas GNU/Linux i686 y X86_64 como instalador independiente para Windows y como código fuente para Mac OS X [14].

1.2.2. Step

Step es un software interactivo que permite la realización de simulaciones de física en 2D. Diseñado y creado por el físico Vladimir Kuznetsov, el software funciona de la siguiente manera: se añaden cuerpos y otros elementos dentro de un escenario común a los cuales se agregan algunas características especiales como fuerzas y demás efectos que inciden en el comportamiento de estos elementos. Cuando el sistema está implementado se pulsa un botón de simulación y Step inmediatamente muestra la evolución del escenario construido de acuerdo con las leyes de la física. El software permite variar la propiedad de los cuerpos y de las fuerzas involucradas en el experimento, incluso en el transcurso de la simulación, con el fin de obtener los resultados respectivos que faciliten la comprensión de los eventos ocurridos [15].

Step es una herramienta educativa de código abierto centrada en la enseñanza y experimentación de un gran número de fenómenos comunes en las diferentes

ramas de la física, entre las que se encuentran la física mecánica, la electrostática, la termodinámica y la dinámica molecular. Dichos fenómenos son desarrollados bajo secuencias numéricas y gráficas. Step es un software libre funcional desarrollado a partir de motores físicos incluidos en el escritorio KDE, es decir, un conjunto de aplicaciones e infraestructura de desarrollo disponible para un variado número de sistemas operativos tales como GNU/Linux, Mac OS X, Windows, entre otros [16].

1.2.3. Algodoo

Es un software abierto de simulación de física en dos dimensiones, el cual facilita la creación de múltiples escenarios a partir de una gran variedad de componentes disponibles dentro de su interfaz. Los sistemas físicos o escenarios de simulación se implementan por medio de herramientas intuitivas que permiten dibujar elementos tales como polígonos, círculos, engranajes, cuerdas, ejes, motores, entre otros. Algodoo brinda al usuario la posibilidad de interactuar con el sistema creado partiendo de la configuración de parámetros iniciales de los componentes incluidos en el escenario, de esta manera al ejecutar una simulación todos estos elementos actúan regidos por las leyes de la física aplicables dentro de cada modelo [17].

Algodoo es una herramienta que se ejecuta en sistemas operativos como Windows y Mac OS X, también brinda soporte para su utilización en dispositivos móviles como tabletas y Smartphones. Además cuenta con una gran comunidad de respaldo en términos de desarrollo y usuarios activos para compartir todo tipo de escenas de simulación, las cuales incluyen temáticas diversas enfocadas en las ramas de la física, entre ellas la mecánica. El software fue creado a partir del lenguaje de scripting denominado Thyme, el cual es puesto a disposición del usuario para programar comportamientos especializados en los componentes de un escenario [18].

1.2.4. Tracker

Tracker es un programa gratuito de análisis de video y construcción de modelos desarrollado a partir del lenguaje Java, como parte del proyecto Open Source Physics (OSP, Física de Código Abierto), utilizado en la enseñanza de la física. Gracias a la técnica de modelación en video, por medio de Tracker se pueden combinar videos y modelación por computadora para construir escenarios con sistemas físicos diversos, apegados al mundo real, como por ejemplo sistemas de caída libre, de movimiento sobre planos inclinados, de movimiento circular, movimiento parabólico, de equilibrio de cuerpos, entre otros.

Algunas de las características de Tracker incluyen el seguimiento manual y automático de objetos con superposición de la posición, velocidad y aceleración, seguimiento del centro de masa, gráficos de vectores y suma de segmentos orientados de manera interactiva, entre otros. Por medio de la opción Model Builder se pueden crear modelos dinámicos y cinemáticos completos de partículas de masa definida y sistemas de dos o más cuerpos. Por su parte las superposiciones de modelos se sincronizan automáticamente y se escalan al video para una comparación directa con el mundo real.

En cuanto a la generación de datos para llevar a cabo los diferentes análisis, Tracker dispone de algunos elementos tales como escalas fijas o variables en el tiempo para los sistemas de coordenadas, origen e inclinación, diferentes tipos de herramientas para medir valores de distancia y ángulos, herramientas para el ajuste de curvas de forma manual o automática, despliegue de valores medidos utilizando formatos numéricos ajustables, entre otros [19].

1.2.5. Interactive physics

Interactive Physics es un software educativo perteneciente a la empresa de desarrollo Design Simulation Technologies, el cual permite explorar, descubrir y observar el mundo físico por medio de la simulación de sistemas y escenarios construidos por los usuarios a partir de una amplia selección de controles, componentes, objetos, parámetros y ambientes disponibles en la interfaz de la herramienta.

De acuerdo a los elementos añadidos dentro del escenario implementado, el programa permite incorporar a estos algunos efectos de contacto, colisiones, fricción, alteración de los parámetros gravitatorios y de fricción del aire, entre otros. También se pueden realizar diferentes tipos de mediciones dentro de cada sistema, como por ejemplo valores de velocidad, aceleración y energía de cada uno de los objetos incluidos. Actualmente el software está disponible para las plataformas de Windows y MAC OS X y puede ser adquirido por medio del pago de una licencia [20].

Interactive Physics abarca un gran número de temáticas relacionadas con las ramas de la física, entre ellas la correspondiente a la física mecánica, de la cual es posible llevar a cabo las siguientes simulaciones: resistencia del aire, colisiones, leyes de conservación de la energía, equilibrio, marcos de referencia inerciales, fricción, momento, movimiento en 1 y 2 dimensiones, trabajo y energía, dinámica de partículas, movimiento sobre planos, sistemas de poleas, dinámica rotacional, vectores y leyes de Newton [21].

1.2.6. Physion

Physion es un software de simulación de física en 2D desarrollado por Dimitris Xanthopoulos en el año 2010, mediante el cual se pueden crear una amplia gama de simulaciones físicas interactivas y otros experimentos educativos de manera sencilla. Dadas sus características, Physion es una herramienta útil para profesores y alumnos, ya que puede ser utilizada como un laboratorio virtual para la enseñanza y comprobación de algunos de los conceptos más importantes relacionados con las ramas de la física.

Physion dispone de una amplia gama de herramientas para crear diferentes objetos físicos como círculos, polígonos y engranajes, además de articulaciones como por ejemplo resortes, cuerdas, poleas, entre otros, cuyo funcionamiento estará determinado por las leyes de la física. Con estos elementos el usuario puede crear infinidad de escenarios posibles para llevar a cabo experimentos de física y ejecutar otros sistemas que hayan sido diseñados y creados por medio de scripts (programas en lenguaje JavaScript) [22].

Physion está desarrollado básicamente a partir de dos grandes librerías: Qt SDK, que corresponde a un marco multiplataforma integral con herramientas para la creación de aplicaciones nativas e interfaces de usuario para diferentes tipos de dispositivos [23], y Box2D, que es un motor de física de código abierto para la simulación de cuerpos rígidos en 2D [24]. El software está disponible para los sistemas operativos Windows (XP, Vista, 7) y Linux (x86 y x86_64) y su distribución es completamente gratuita [25].

1.2.7. Cienytec (Laboratorio de física virtual)

Es un software creado y distribuido por CIENYTEC, empresa líder en el suministro de instrumentos científicos y equipos para laboratorios en Centro América, el Caribe y Sur América. Por medio del laboratorio virtual se puede llevar a cabo la simulación de experimentos físicos de manera sencilla, con la obtención de resultados numéricos y gráficos confiables. El sistema permite a los docentes el acompañamiento, supervisión y control del trabajo de los estudiantes en tiempo real a través de herramienta de seguimiento dentro del aula de clase.

El software pone a disposición herramientas necesarias que facilitan la labor del docente en cuanto a actividades para complementar los contenidos de los cursos por medio de material externo proveniente de diversas fuentes, envío de videos y otros archivos a los estudiantes, recepción de documentos y resultados de las simulaciones realizadas por los alumnos, entre otros. La aplicación incluye diferentes barras de comandos con algunas funciones como por ejemplo: Archivo

(nuevo, abrir, guardar, imprimir, configurar página, archivos recientes y salir), Editar (deshacer, rehacer, cortar, copiar, pegar, seleccionar todo, eliminar, entre otros), Ver (panel lateral, barra de herramientas, pantalla completa, zoom, etc), Escena (pantalla donde el usuario puede montar los diferentes escenarios de simulación), Ayuda, entre otras.

En cuanto a las temáticas tratadas, relacionadas con la física mecánica, el laboratorio virtual Cienytec permite llevar a cabo las siguientes simulaciones: Descripción del movimiento, fuerza y aceleración, energía y movimiento. El software ha sido diseñado para los sistemas operativos Windows (2000, XP y posteriores) y Linux (Kernel 2.6 o posterior) y está disponible en idioma español [26].

1.2.8. Phet interactive simulations

Es un proyecto de simulaciones interactivas de matemáticas y ciencias fundado en el año 2002 por el ganador del premio nobel de física Carl Wieman. La plataforma Phet pone a disposición del público una gran cantidad de simulaciones de forma gratuita, compuestas por sistemas de diferentes características implementados dentro de un ambiente intuitivo, similar a un juego, lo que permite complementar de manera didáctica los conceptos teóricos asociados a cada experimento simulado [27].

Las simulaciones de Phet se desarrollan bajo los siguientes principios de diseño: inclusión de múltiples representaciones a nivel de la interfaz visual (movimiento de objetos, despliegue de gráficos, resultados numéricos, entre otros), sistemas apegados al mundo real, interactividad por medio de elementos que permiten modificar las características de simulación de los escenarios implementados, sistemas flexibles que puedan ser utilizados en diferentes contextos educativos, inmediatez en los resultados obtenidos como efecto del cambio en las variables de entrada dentro de cada sistema, entre otros [28].

Dentro de las simulaciones disponibles en la plataforma Phet se destacan aquellas destinadas al estudio de la física mecánica, entre las que se encuentran: Ley de Hooke, fuerza y movimiento (fundamentos), gravedad y órbitas, energía en la pista de patinaje (conceptos básicos), ley de equilibrio, laboratorio de resortes y masa, pista de patinar (energía), laboratorio de péndulo, laboratorio de colisiones, torsión, movimiento de un proyectil, fricción, fuerzas en 1 dimensión, movimiento de mariquita en 2D, laboratorio de fuerza de gravedad, Phet movimiento en 2D y fuerzas y movimiento dentro de una rampa [29].

En cuanto a las herramientas utilizadas para la implementación de estas simulaciones se cuentan con tres lenguajes principales: el lenguaje Java (para el desarrollo de *applets* en la web), el lenguaje Action Script (plataforma Adobe Flash) y el lenguaje de desarrollo web HTML5. Inicialmente el proyecto Phet desarrolló cada una de sus simulaciones en los lenguajes Java y Action Script, los cuales requerían para su funcionamiento (por medio de navegadores web) de elementos adicionales como por ejemplo el JRE o máquina de virtual de Java y el plugin Macromedia Flash Player respectivamente.

Sin embargo, desde hace algunos años, y como consecuencia de la no continuidad en el soporte a este tipo de elementos accesorios por parte de los exploradores web más conocidos como Google Chrome o Mozilla Firefox, el proyecto Phet ha empezado a desarrollar todas sus simulaciones con el lenguaje web HTML5, lo que ha permitido también su despliegue dentro de los dispositivos móviles como tabletas y teléfonos inteligentes. En cuanto a su funcionalidad, actualmente las simulaciones de Phet están disponibles de manera gratuita en su página web oficial, con la posibilidad de ser descargadas si se quiere trabajar con ellas en sitios donde no haya acceso a la misma [30].

1.2.9. Fislab.net

Es un laboratorio virtual de física compuesto por *applets* (simulaciones de sistemas físicos desplegados a través de una página web), apuntes (elementos teóricos referentes a los temas físicos tratados en las simulaciones) y ejercicios (listado de problemas y cuestionarios agrupados temáticamente) disponibles a través de su sitio web oficial. El conjunto de simulaciones, desarrolladas en lenguaje Java, comprenden tres grandes temáticas relacionadas con la física mecánica, entre ellas se tienen: caída libre, movimiento rectilíneo de 1 y 2 cuerpos, movimiento parabólico y movimiento circular, correspondientes al tema de cinemática.

También se encuentran disponibles simulaciones de movimiento sobre planos inclinados, movimiento de dos objetos sometidos a fuerzas, colisiones y momento de inercia, correspondientes al tema de dinámica. El material contenido en la página web es de libre difusión, por lo tanto se tiene acceso a los ficheros JAR (Java ARchive) y a los archivos HTML que contienen los *applets* (simulaciones) para ser modificados y difundidos en otras sitios web, siempre y cuando se haga referencia al dominio Fislab.net como propietaria intelectual de este conjunto de simulaciones [31].

1.2.10. HTML5: physics lab simulations

Corresponde a una vasta colección de simulaciones gratuitas, desarrolladas en lenguaje HTML5 (HTML, JavaScript y CSS3) y puestas a disposición de profesores y alumnos para el estudio y experimentación virtual de algunos de los aspectos más importantes relacionados con la física. Gracias al desarrollo en HTML5, las simulaciones pueden ser desplegadas en dispositivos que van desde teléfonos móviles inteligentes y tabletas hasta ordenadores de escritorio.

La aplicación web cuenta con simulaciones que abarcan un gran número de temas relacionados con la física en general (mecánica, física de fluidos, térmica, electricidad y magnetismo, vibraciones y ondas, óptica y física moderna). En cuanto a las simulaciones disponibles para explorar los fenómenos que hacen parte de la física mecánica se tienen los siguientes experimentos virtuales: laboratorio de movimiento, ecuaciones de movimiento, práctica de aceleración, representación gráfica de movimiento simple, caída libre, aceleración debida a la gravedad, segunda ley de Newton, laboratorio de fricción, fuerza sobre un plano inclinado, movimiento circular, transformación de energía, momento de inercia, entre otros. Las simulaciones se encuentran disponibles en el sitio web oficial del proyecto [32].

1.2.11. Physics interactives

Es una colección de simulaciones interactivas gratuitas con las que los usuarios pueden explorar algunos de los conceptos más importantes de la física en general. Dentro de cada simulación se da la opción de manipular el entorno implementado con el fin de observar los cambios producidos por la variación de dichos parámetros iniciales. Las simulaciones vienen acompañadas por una hoja de actividades con una serie de pasos predefinidos que sirven como complemento didáctico dentro de cada uno de los experimentos virtuales.

La plataforma dispone de una serie de simulaciones enfocadas en la física mecánica, las cuales comprenden las siguientes temáticas: cinemática en 1 dimensión, leyes del movimiento de Newton, vectores, proyectiles y movimiento en dos dimensiones, momento y colisiones, trabajo y energía, movimiento circular y gravitación, así como también un sistema dedicado al tema de equilibrio y rotación. Las diferentes simulaciones han sido desarrolladas en lenguaje HTML5 (HTML, JavaScript y CSS3) y se encuentran disponibles en su sitio web oficial [33]:

1.2.12. Animaciones de física (universidad politécnica de Madrid)

Corresponden a un conjunto de experimentos basados en animaciones gráficas en 2 dimensiones desarrolladas en la plataforma Adobe Flash (lenguaje Action Script) por Teresa Martín Blas y Ana Serrano Fernández, docentes de física de la Universidad Politécnica de Madrid (UPM). Las simulaciones abarcan diferentes temáticas relacionadas con la física en general, entre ellas algunas referentes a la física mecánica, entre las que se encuentran las siguientes animaciones: componentes intrínsecas de la aceleración, tiro parabólico, movimiento relativo de traslación, movimiento relativo de rotación uniforme, correspondientes al tema de cinemática.

Por otra parte, también están disponibles simulaciones de trayectorias de cuerpos, rozamiento con el aire, par acción-reacción, plano inclinado con rozamiento, movimiento armónico simple y fuerzas de inercia correspondientes al tema de dinámica de una partícula. Finalmente, como parte de la temática de dinámica de sistemas de partículas se tienen las simulaciones de colisiones elásticas y de fuerzas internas y externas actuando sobre un sistema definido de cuerpos. Todas las simulaciones están disponibles de manera gratuita en el sitio web del proyecto [34]. Las simulaciones cuentan con un material de apoyo correspondiente al resumen de conceptos teóricos, cuestionarios y solucionario de los problemas propuestos.

1.3. CARACTERISTICAS DE IMPLEMENTACIÓN

Las ventajas citadas en el apartado 1.1 respecto a la importancia de las herramientas de simulación en la enseñanza de la física son generales y se aplican a todas las ramas que hacen parte de esta ciencia, incluyendo la mecánica. Por lo tanto, teniendo claros los aspectos relevantes que potencializan el uso del software como apoyo pedagógico en la enseñanza de la mecánica y con base en las propiedades de las herramientas de simulación de la sección 1.2 se resaltan las características más importantes que debe cumplir una aplicación como elemento didáctico para su utilización dentro de un entorno educativo específico. En la tabla 1 se resumen algunas de estas características [35]:

Tabla 1. Características de implementación de la herramienta de simulación.

Característica	Definición
▪ Requerimientos de sistema razonables.	La implementación de la herramienta en lo posible no debe requerir grandes recursos de hardware y software.

▪ Facilidad de instalación	Pasos bien definidos, consignados dentro de un manual de instalación.
▪ Facilidad de uso	La utilización de la herramienta por parte de los estudiantes debe ser lo más intuitiva posible.
▪ Interactividad	El software debe brindar una respuesta rápida a las peticiones o acciones de los estudiantes de manera que el intercambio de información sea lo más transparente posible.
▪ Calidad en el entorno de comunicación	Es decir, que la aplicación muestre con claridad al usuario cada uno de los sistemas físicos implementados a través de gráficos bien definidos que permitan la identificación de los componentes importantes asociados a ellos.
▪ Calidad en el entorno audiovisual	La aplicación debe contar con un diseño atractivo, basado en la utilización de elementos gráficos tales como títulos, menús, ventanas, iconos, botones, cuadros de texto, cuadros de imágenes, gráficas y cuadros de resultados, entre otros.
▪ Calidad en los contenidos	Información correcta y actualizada, que sea clara para los usuarios.
▪ Parametrizable	Que permita la modificación de los parámetros importantes dentro de cada sistema de simulación, como por ejemplo: valores de las masas de los cuerpos, coeficientes de fricción de las superficies, distancias, valores de posición y velocidad inicial de las partículas, entre otros.
▪ Accesibilidad	Es decir, que brinde al estudiante la facilidad de trabajar con las simulaciones en cualquier momento y lugar, siempre y cuando existan las condiciones necesarias para su utilización.

1.4. HERRAMIENTAS DE DESARROLLO

En este punto se realiza una delimitación inicial explorando las ventajas y desventajas más representativas de los tres tipos de herramientas implementadas como plataformas para la realización de prácticas virtuales de física mecánica (sección 1.2) y partiendo de estas propiedades se establece la opción que más se acomode a las características de implementación dadas en el numeral 1.3. De esta manera, los tres tipos de aplicación puestos en consideración son los siguientes:

- Aplicaciones de escritorio.
- Aplicaciones web.
- Aplicaciones móviles.

1.4.1. Aplicaciones de escritorio

Son herramientas creadas para ser ejecutadas en un ordenador de escritorio a través de un sistema operativo de interfaz visual como Windows o Linux. Para su funcionamiento no requieren de una conexión a un servidor y dependen enteramente de las características del sistema operativo para el cual fue desarrollada [36]. La tabla 2 muestra algunas de las características más importantes que presentan este tipo de herramientas [37]**:**

Tabla 2. Ventajas y desventajas de las aplicaciones de escritorio.

Ventajas	Desventajas
▪ En su mayoría suelen ser mucho más robustas y estables que las aplicaciones web. ▪ En cuanto al rendimiento, el tiempo de respuesta a las peticiones del usuario es muy rápido. ▪ Dependiendo de las personas a cargo del desarrollo pueden llegar a ser muy seguras. ▪ Dado que su ejecución se lleva a cabo de manera local, la comunicación con el exterior no es relevante. Esta característica deriva	▪ El acceso a la aplicación se limita solo al equipo donde está instalado. ▪ Requieren instalación y actualización personalizada. ▪ Por lo general requieren características especiales de software y librerías. ▪ Dependen del sistema operativo que utiliza el ordenador y sus prestaciones están limitadas a las capacidades de este (memoria, tarjeta de video, entre otras.)

en una mayor velocidad de procesamiento, lo que impulsa al desarrollo de herramientas más potentes y funcionales.	

1.4.2. Aplicaciones web

Son herramientas cuya ejecución y funcionamiento se realizan enteramente a través de internet, de esta manera los datos y archivos que las componen se procesan y son almacenados dentro de la web [38]. La información contenida en estas aplicaciones se guarda de forma permanente en servidores web y se pone a disposición de los usuarios a través de un navegador o explorador de internet. Las ventajas y desventajas más importantes de este tipo de aplicaciones se muestran en la tabla 3 [39]:

Tabla 3. Ventajas y desventajas de las aplicaciones web.

Ventajas	Desventajas
▪ Compatibilidad multiplataforma gracias a que operan a través de un navegador web, lo que permite que se puedan utilizar desde cualquier dispositivo, sin importar el sistema operativo. ▪ Requieren una capacidad menor de memoria RAM por parte del equipo del usuario final. ▪ Brindan acceso inmediato a la aplicación ya que no requieren de pasos previos de instalación y configuración. Solamente se necesita un explorador web y una conexión a internet para trabajar con dicha aplicación. ▪ Las aplicaciones basadas en web siempre están actualizadas con la última versión del software.	▪ La aplicación no funciona si no existe conexión a una red de internet. ▪ El usuario final por lo general no tiene la opción de elegir cual versión de la aplicación es la que quiere utilizar. ▪ El tiempo de respuesta por parte de la aplicación puede ser más lento si se compara por ejemplo con una herramienta de escritorio. ▪ En algunas ocasionas se ven afectadas por las actualizaciones del navegador web.

▪ Soportan multiplicidad de usuarios trabajando al mismo tiempo en la aplicación. ▪ Todos los usuarios utilizan la misma versión de la aplicación. ▪ Facilita el trabajo a distancia entre los miembros de un mismo grupo.	

1.4.3. Aplicaciones móviles

Una aplicación móvil es una herramienta informática desarrollada para ser ejecutada en dispositivos móviles inteligentes tales como teléfonos celulares, tabletas, entre otros. Son desarrolladas para distintas plataformas dependiendo del fabricante y su distribución se realiza a través de tiendas virtuales, en donde pueden ser adquiridas de forma gratuita o a un precio determinado [40]. Las principales ventajas y desventajas de este tipo de aplicaciones se muestran en la tabla 4 [41]:

Tabla 4. Ventajas y desventajas de las aplicaciones móviles.

Ventajas	Desventajas
▪ En su mayoría, no requieren de una conexión a internet para su funcionamiento. ▪ Simplicidad de uso. ▪ Inmediatez en el acceso a la aplicación debido a que se instalan sobre el dispositivo móvil.	▪ Las aplicaciones no se pueden descargar en todos los dispositivos móviles. ▪ Solo pueden ser utilizadas por los dispositivos que cuenten con el mismo sistema para la cual fue desarrollada. ▪ Dado que las aplicaciones van instaladas en los dispositivos móviles, al existir una actualización se debe reinstalar la aplicación, lo que se hace que se mantengan las versiones antiguas del software.

	▪ Desarrollo limitado por las plataformas de los dispositivos. ▪ Código personalizado y exclusivo. Es decir que si se quiere desarrollar una aplicación nativa en tres plataformas móviles distintas, hay que hacer el desarrollo una por una en su respectivo lenguaje de programación (no se puede reutilizar código). ▪ Pueden requerir aprobación para ser publicadas en la plataforma.

1.4.4. Elección del tipo de aplicación

De acuerdo a las características de implementación dadas en la sección 1.3 y a las propiedades de los tres tipos de aplicaciones disponibles, resumidas en las tablas 2, 3 y 4 respectivamente, se escoge llevar a cabo el desarrollo de una aplicación web básicamente por las siguientes razones:

- **Personalización, actualización y soporte de la herramienta:** para todos los usuarios se podrá hacer realizando únicamente los cambios pertinentes a nivel de servidor web.

- **Accesibilidad y cobertura:** los usuarios podrán hacer uso de la aplicación a cualquier hora y desde cualquier lugar donde haya acceso a internet.

- **Portabilidad:** la aplicación podrá ser usada con cualquiera de los navegadores más comunes en la web actual.

- **Infraestructura y movilidad:** los usuarios solo deberán estar a conectados a internet para hacer uso de la herramienta de simulación.

- **Capacidad de usuarios:** alta, debido a que la aplicación estará hecha para soportar un buen número de ellos conectados a la vez.

- **Modos de acceso:** al ser una aplicación de tipo web, los usuarios podrán acceder a ella través de ordenadores y dispositivos móviles.

Una vez escogido el tipo de aplicación que se desea realizar se procede a dar un repaso por algunos de los lenguajes de desarrollo web que proporcionen las herramientas necesarias que permitan la implementación del programa de simulación de mecánica.

1.5. LENGUAJES DE DESAROLLO WEB

Con base en los lenguajes de desarrollo web utilizados en algunas de las plataformas estudiadas en la sección 1.2, y partiendo de la necesidad de contar con un lenguaje que facilite la implementación de una aplicación cuyo funcionamiento esté basado en animaciones gráficas en dos dimensiones que representen ciertos fenómenos y sistemas propios de la física mecánica, se proponen las siguientes herramientas de desarrollo:

- Lenguaje HTML5 (HTML + JavaScript + CSS3).
- Lenguaje ActionScript (Plataforma Adobe Flash).
- Lenguaje Java (Applets).

1.5.1. HTML5 (HTML 5, JavaScript y CSS3)

Al hablar de HTML5 no se está haciendo referencia a una nueva versión del lenguaje de etiquetas HTML (HyperText Markup Languaje) sino a un compendio de distintas especificaciones encaminadas a un mismo propósito: el desarrollo web. HTML5 surge como un nuevo concepto mediante el cual se busca mejorar la construcción de sitios web y aplicaciones que se acomoden a las nuevas exigencias del mercado, en donde los dispositivos móviles, los trabajos en red y la computación en la nube son los elementos que predominan en la actualidad [42].

En términos generales HTML5 permite la diferenciación de cada uno de sus componentes (HTML, CSS3 y JavaScript) en cuanto a funcionalidad y brinda los estándares necesarios para muchos aspectos relacionados con la creación de sitios web. Individualmente los elementos que componen el HTML5 aportan las siguientes funciones: el lenguaje de etiquetas HTML, al igual que sus predecesores, pone a disposición un modelo estructural para los sitios web, CSS3 (Cascading Style Sheets) se encarga de darle el aspecto atractivo y utilizable a dicha estructura y JavaScript otorga dinamismo y funcionalidad a los sitios web creados.

HTML5, CSS3 y JavaScript son estándares web abiertos, fiables y altamente eficientes, compatibles con la gran mayoría de dispositivos utilizados en la actualidad. Estos estándares permiten a los desarrolladores (entre otras cosas) la creación de gráficos avanzados y sistemas basados en secuencias de animación gracias a la inclusión de nuevos elementos como el API Canvas y un gran número de librerías y Frameworks gratuitos disponibles para la implementación de videojuegos [43], recursos que pueden llegar a de mucha utilidad dentro del presente proyecto.

Canvas es un nuevo elemento incorporado dentro de la especificación HTML5 que permite la generación de gráficos estáticos y animaciones a partir de scripts. Como objeto, Canvas es accedido a través de JavaScript para la implementación de gráficos en 2D, juegos y composición de imágenes. Las nuevas versiones de los exploradores ya soportan el elemento Canvas y en la actualidad se cuenta con infinidad de APIs para generar los diferentes gráficos a partir de este elemento [44].

En cuanto a los motores gratuitos para la realización de videojuegos y escenarios de animación a partir de secuencias de imágenes, HTML5 cuenta con un gran número de estos elementos. Algunas de las librerías y Frameworks disponibles para tales propósitos son los siguientes [45]:

- **Create.js:** Es un conjunto de varias librerías escritas en JavaScript utilizadas en el desarrollo de contenido interactivo con HTML5. Se compone de varios módulos, entre ellos uno denominado Easel.js, el cual brinda las herramientas para trabajar con gráficos e interactuar directamente con el elemento Canvas gracias a una API, similar a la de Flash, que utiliza las ventajas de JavaScript.

- **Quintus:** Es un Framework para el desarrollo de juegos en 2D para escritorio y dispositivos móviles que utiliza elementos HTML5. Posee módulos para cargar posteriormente los archivos que serán utilizados dentro del videojuego, tales como imágenes, ficheros JSON o ficheros TMX entre otros. Es un módulo ideal para crear escenarios y todo tipo de escenas de animación.

- **Phaser.js:** Es un potente framework de código abierto que permite la creación de juegos en 2D para navegadores web haciendo uso de elementos HTML5, los cuales pueden ser desplegados en ordenadores y dispositivos móviles por igual. Soporta Canvas y WebGL y puede hacer uso de cada uno indistintamente, apoyándose en las características que brinda el navegador. Tiene soporte para Smartphones y tabletas y permite la entrada/salida de datos a través de diferentes periféricos como teclado, mouse, multitouch, entre otros. Cuenta con códecs para archivos de audio y brinda soporte para el manejo de sprites y creación de secuencias de animación. Phaser es un Framework que está en

continuo desarrollo por lo que se disponen de nuevas versiones con bastante frecuencia.

- **Cocos 2D:** Es una librería conformada por un conjunto de herramientas para el desarrollo de videojuegos en diferentes plataformas como Windows, Linux y OS X. Está compuesta por una serie de módulos con diferentes características, entre ellos se tienen: Cocos2d-x, un framework para desarrollar videojuegos y otras aplicaciones interactivas de código libre escritos en lenguaje C++, Cocos2d-JS, un módulo para desarrollo de videojuegos en JavaScript, cocos2d-XNA, un framework de desarrollo de videojuegos en dos y tres dimensiones escrito en C#, Cocos2d (Python), un Framework para desarrollo de aplicaciones en lenguaje Python, entre otros. Cuenta también con una serie de APIs para el manejo de sprites, animaciones, partículas, eventos y otros elementos multimedia.

1.5.2. Actionscript (Adobe Flash)

La plataforma Flash se compone de un conjunto de tecnologías utilizadas para la creación de contenidos interactivos multimedia desplegados dentro de aplicaciones para la web o aplicaciones nativas de escritorio. Gracias al lenguaje Actionscript se pueden desarrollar diferentes aplicaciones con contenido provisto de animaciones para distintos medios como equipos de cómputo y aplicaciones para dispositivos móviles no basadas en la web. Algunas de las funciones que Flash permite realizar son las siguientes [46]:

- **Creación de imágenes:** Esta función permite exportar imágenes estáticas a partir de animaciones creadas con Flash en diferentes formatos tales como JPEG, GIF, BMP, PICT, entre otros.

- **Documentos de Flash:** Conformado a su vez por un documento padre que le permite componer una escena de animación a base de dibujos. Esta escena, al ser exportada como una película puede ser reproducida a través de un reproductor de video indicado como por ejemplo el flash player o reproductor de flash. El formato del documento padre es .FLA, mientras que la extensión de la película exportada es .SWF.

- **Animaciones:** Flash provee las herramientas necesarias para la creación de variados tipos de animaciones con base en la integración de un conjunto de

imágenes que generan una determinada secuencia dentro de un lapso de tiempo definido.

- **Juegos:** El lenguaje Actionscript permite la creación de juegos y de un gran número de aplicaciones interactivas sin la necesidad de contar con amplios conocimientos en programación. Para desarrollar un juego con Flash se disponen de varias herramientas dentro del panel de edición Actionscript al igual que acciones automatizadas, previamente definidas, que simplifican el trabajo.

- **Sitios web:** Durante un buen tiempo Adobe Flash se convirtió en la plataforma más utilizada dentro del desarrollo de aplicaciones de internet enriquecidas (RIA) gracias a su enfoque en animación y video. Sin embargo su éxito y popularidad se fueron aminorando con el paso de los años como consecuencia del uso de tecnologías accesorias para su funcionamiento, elementos a los cuales los exploradores web han ido dejando sin soporte [47].

1.5.3. Java (Applets en la web)

El lenguaje Java permite el desarrollo de múltiples y variadas aplicaciones. Una de ellas son las denominadas aplicaciones *standalone*, las cuales hacen referencia a programas que básicamente son ejecutados desde la línea de comandos del sistema operativo. La otra son los *servlets*, es decir, programas diseñados para trabajar del lado del servidor, con los que se puede desarrollar un gran número de aplicaciones de tipo web para la interacción con los clientes [48]. Finalmente están los *applets*, los cuales se detallan a continuación:

Los *applets* son pequeñas aplicaciones escritas en lenguaje java y que de acuerdo a su funcionalidad son incluidas dentro de algunas páginas web creadas a partir de HTML. Los *applets* se pueden ejecutar con cualquier navegador que incorpore dentro de sus herramientas un intérprete java, y su ejecución se debe realizar dentro de un elemento contenedor, provisto por un programa anfitrión a través de un plugin [49].

Los *applets* se insertan en las páginas web, al igual que las animaciones, videos u otros objetos de la plataforma Flash, haciendo uso de una etiqueta especial para ello. Estos elementos se afectan directamente con las reglas de la página web que los aloja, a pesar de ser programas independientes. Para su funcionamiento el navegador debe cargar la página web, de esta manera el *applet* insertado también se carga y ejecuta sus funciones [50].

Una de las ventajas que presentan los *applets* es que son elementos multiplataforma ya que funcionan como aplicaciones dentro de cualquier sistema operativo en el que exista una máquina virtual Java (Linux, Windows, Mac OS, entre otros). Sin embargo, una gran desventaja es que requieren del plugin de Java, el cual no se encuentra disponible por defecto dentro de todos los exploradores web [51].

1.6. COMPARACIÓN DE LOS LENGUAJES DE DESARROLLO WEB

Después de repasar algunas de las características más importantes de los lenguajes opcionales para llevar a cabo el desarrollo web, se realiza una comparativa con base en ciertas propiedades importantes desde el punto de vista del desarrollador con el fin de esclarecer el panorama que conlleve a una buena decisión en la escogencia de dicho lenguaje (ver tabla 5) [52].

Tabla 5. Características y comparación de los lenguajes de desarrollo web.

	JavaScript (HTML5)	**ActionScript (Flash)**	**Java (applets)**
Facilidad de aprendizaje	Media	Media	Media
Páginas dinámicas	Si	Si	Si
Estructura del lenguaje	Scripting	POO	POO
Facilidad de diseño	Media	Media	Media
Licencia	Libre	Propietaria	Libre
Conexión a BD	No	No	No
Lugar de ejecución	Cliente	Cliente	Cliente
Requisitos	Explorador web	Explorador web + Plugin Macromedia Flash Player	Explorador web + Plugin JRE de Java
Compatibilidad	Multiplataforma	No compatible con los navegadores de los dispositivos móviles	No compatible con los navegadores de los dispositivos móviles

1.6.1. Elección del lenguaje de desarrollo web

De acuerdo a las propiedades de cada lenguaje de programación dadas en la sección 1.5 y a la comparativa realizada en la tabla 5 se llega a la conclusión de que HTML5 es la herramienta que mejor se acomoda al desarrollo de la aplicación web requerida dentro del presente trabajo de grado. Las razones principales por las que se escoge este lenguaje son las siguientes:

- El trabajo con el lenguaje HTML5 permite llevar a cabo un desarrollo rápido, con una gran disminución en las líneas de código y un ahorro importante de tiempo comparado con otros lenguajes de programación, además de que aporta grandes ventajas en términos de seguridad y de construcción de aplicaciones cada vez más dinámicas, todo ello gracias al equipo de trabajo que está detrás del mantenimiento y progreso del estándar de desarrollo web [53].

- Desde su nacimiento la especificación HTML5 ha sido bien acogida por diseñadores de páginas web y por los fabricantes de aplicaciones gracias a las características y posibilidades que ofrece, y desde el 2014, año en el la especificación pasó a ser un estándar oficial, se viene posicionando como uno de los más utilizados. Además es un lenguaje compatible con múltiples sistemas operativos de diferentes plataformas, lo que permite producir experiencias a través del navegador que antes solo eran posibles con la ayuda de software de adicional, como en el caso de Flash, Java en la web y Silverlight [54].

- Grandes empresas como Microsoft, Google, Apple, Facebook, Yahoo, Mozilla y otros proyectos tecnológicos, independientemente de su naturaleza, apoyan y utilizan HTML5 como pieza clave en algunas de las más importantes estrategias de posicionamiento comercial, aspecto que lo convierten en el presente y futuro de la web [43].

- HTML5 viene a ser el sucesor de Flash dentro del desarrollo de Apps para una tecnología web que siempre va más allá de las habilidades de un navegador: audio, video, webcams, animaciones en 2 y tres dimensiones, componentes de interfaz complejas, entre otras. Gracias a su naturaleza, HTML5 es capaz de hacer todo esto sin la necesidad de plugins y con gran compatibilidad en casi todos los navegadores actuales [43].

- En la actualidad el uso de la tecnología Flash se ha visto limitado en gran medida por las restricciones en los navegadores de los dispositivos que impiden la instalación del plugin necesario para su funcionamiento. En este aspecto los elementos gráficos de HTML5, como el caso del API Canvas, juegan un papel muy importante, ya que aparte de funcionar en diferentes plataformas, ofrecen una potencia de visualización bastante equiparable a lo ofrecido por Flash hasta

el momento y sin ningún problema de funcionamiento dentro de estos dispositivos, ya que pueden hacerlo a través de cualquier navegador web que soporte HTML5 (Google Chrome, Mozilla Firefox, Opera, Safari, entre otros) [44].

- HTML5 cuenta con una gran cantidad de bibliotecas y demás recursos gratuitos para el desarrollo de aplicaciones de diferente naturaleza (entre ellos Frameworks y motores de videojuegos para la implementación de aplicaciones basadas en animaciones), con una muy buena documentación de apoyo y un apartado con infinidad de ejemplos que permiten el acceso al código fuente para la comprensión de su funcionamiento. Además tiene una gran comunidad de respaldo que continuamente está mejorando la documentación y creando nuevos ejemplos en los que se abarcan diferentes tópicos de desarrollo [55].

- HTML5 al ser un lenguaje del lado del cliente delega toda la carga de procesamiento al navegador web, por ello el despliegue de las aplicaciones no demandan requerimientos especiales de software por parte del equipo donde se corren las simulaciones y su ejecución es llevada a cabo sin ningún problema [54]. Por el contrario, para correr una aplicación desarrollada en lenguaje Java (applets en la web) el usuario debe instalar un entorno de ejecución apropiado y pasar por varios pasos y diálogos antes de conseguirlo, lo que prolonga más de la cuenta la simulación de cualquier experimento implementado a través de esta tecnología [47]

- El estándar HTML5 supone un gran avance al permitir la visualización del contenido en diferentes dispositivos, además de un ordenador. Esta portabilidad y la positiva proyección de HTML5 por encima de otras tecnologías permiten una larga vida útil y adaptación a futuros avances para los proyectos desarrollados bajo esta especificación [56].

- HTML5 en estos momentos tiene soporte universal en la mayoría de las plataformas, y en algunos casos de manera exclusiva, por lo que en términos de despliegue de contenidos a través de los navegadores web, se puede asegurar que es la mejor opción [57].

Capítulo 2

IMPLEMENTACIÓN DE PRÁCTICAS DE FÍSICA MECÁNICA

En el presente capítulo se aborda el proceso general de construcción de la herramienta de simulación, partiendo de la definición y el diseño de los sistemas físicos hasta llegar a los respectivos algoritmos que definen la funcionalidad de cada una de las prácticas que se van a implementar, los cuales serán la base para el proceso de codificación final.

2.1. METODOLOGÍA DE TRABAJO

Con el fin de dar cumplimiento a todos los objetivos impuestos dentro del presente proyecto se considera necesario adoptar una estrategia de trabajo mediante la cual se lleven a cabo una serie de procesos secuenciales que permitan abordar con éxito el desarrollo de la herramienta de simulación. Para ello se recurre a un modelo de trabajo en cascada como el mostrado en la figura 1, con el fin de definir unas actividades fundamentales que puedan ser administradas y evaluadas dentro de cada una de los subsistemas (prácticas) que conformarán el proyecto total (simulador) [58].

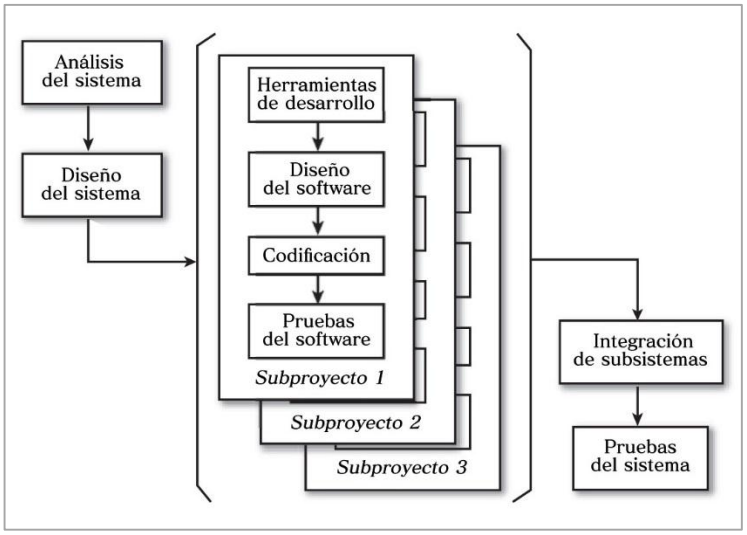

Figura 1. Modelo de trabajo en cascada con subproyectos.

A través del modelo se establecen los pasos fundamentales para llevar a cabo el desarrollo de subproyectos independientes que no dependen entre sí para su funcionamiento. Dichos pasos en esencia son los siguientes:

- **Análisis del sistema:** Hace referencia al procedimiento inicial del proyecto y tiene como objetivo dar una idea general de los requisitos que debe cumplir el sistema y las funcionalidades que se van a ofrecer al usuario. En este apartado se expone lo que se va a implementar y no la manera de realizarlo.

- **Diseño del sistema:** Se formalizan los requerimientos iniciales para cada una de las partes que componen el sistema general, puntualizando en los aspectos y características que debe incluir cada una de las prácticas que se van a implementar a través de la aplicación.

- **Herramientas de desarrollo:** Se determinan los elementos que van a hacer parte del desarrollo del sistema con base en las funcionalidades que ellos aportan a partir de su utilización.

- **Diseño del software:** describe la funcionalidad interna del producto a través de elementos de salida como diagramas de flujo y texto.

- **Codificación:** Se realiza la implementación en código de los algoritmos y estructuras de datos diseñados en etapas anteriores con base en el lenguaje de programación escogido. En este punto se van realizando las primeras pruebas de funcionalidad respecto a los requerimientos dados en el diseño del sistema.

- **Pruebas del software:** El objetivo de esta etapa es garantizar que el programa desarrollado no contenga errores de diseño o de codificación. No se centra en establecer si el producto realiza o no lo estipulado en los requerimientos, solo en encontrar la mayor cantidad de errores.

- **Integración de los subsistemas:** Se integran los componentes que van a formar parte del producto final. En este caso se realiza la unión de las seis prácticas individuales desarrolladas dentro de un mismo entorno.

- **Pruebas del sistema:** Corresponde al paso final de la metodología de trabajo y en él se realizan las pruebas y correcciones finales con base en los elementos ya integrados.

2.2. ANÁLISIS DEL SISTEMA

En término generales la aplicación web que se busca implementar debe incorporar una serie de prácticas individuales relacionadas con algunas de las temáticas del curso de mecánica y su funcionamiento estará basado en secuencias de animación en dos dimensiones que representen de manera coherente la interacción de uno o más cuerpos dentro de un sistema físico definido, donde los parámetros de entrada de dicho sistema se puedan variar. La herramienta permitirá al estudiante experimentar diferentes situaciones relacionadas con cada uno de los sistemas y dará a conocer, por medio de valores numéricos y gráficas de movimiento, los resultados obtenidos en el proceso de simulación.

2.3. DEFINICIÓN DE LAS PRÁCTICAS DE FISICA MECÁNICA

En los cursos de física mecánica de los programas de ingeniería de la Universidad de Cauca se abordan una serie de temáticas generales que en conjunto conforman el componente teórico de la asignatura sobre el cual se llevan a cabo los respectivos procesos de evaluación por parte de los docentes. Algunos de esos temas incluyen el estudio del movimiento de los cuerpos en una y dos dimensiones, el movimiento vectorial, las leyes del movimiento, el movimiento circular, la energía mecánica de un sistema y sus principios de conservación, cantidad de movimiento lineal y colisiones, rotación de objetos rígidos y cantidad de movimiento lineal.

Considerando el listado anterior se proponen las siguientes temáticas para abordar la implementación de las prácticas virtuales de física mecánica dentro de la aplicación web:

- Práctica 1: Vectores.
- Práctica 2: Movimiento en dos dimensiones.
- Práctica 3: Movimiento circular.
- Práctica 4: Leyes del movimiento.
- Práctica 5: Trabajo y energía.
- Práctica 6: Colisiones.

De esta forma se procede al diseño individual de los sistemas físicos sobre los cuales van a estar estructuradas cada una de las prácticas definidas, como se muestra a continuación:

2.4. PRÁCTICA 1 – VECTORES

2.4.1. Diseño del sistema

La práctica de representación vectorial se basa principalmente en la implementación de vectores dentro de un plano cartesiano definido, como se muestra en la figura 2. La construcción de cada segmento orientado se realiza especificando el tipo de representación que va a tener, pues dependiendo del sistema de coordenadas en el que se trabaje, los datos requeridos para dicha implementación serán diferentes. De esta manera, si se quiere representar un vector por medio de coordenadas cartesianas se deben suministrar los valores de x y y correspondientes al punto que denote la posición de algún objeto dentro del plano cartesiano. En cambio, si los datos dados son la magnitud y dirección del vector, su representación va a estar definida por un sistema de coordenadas polares.

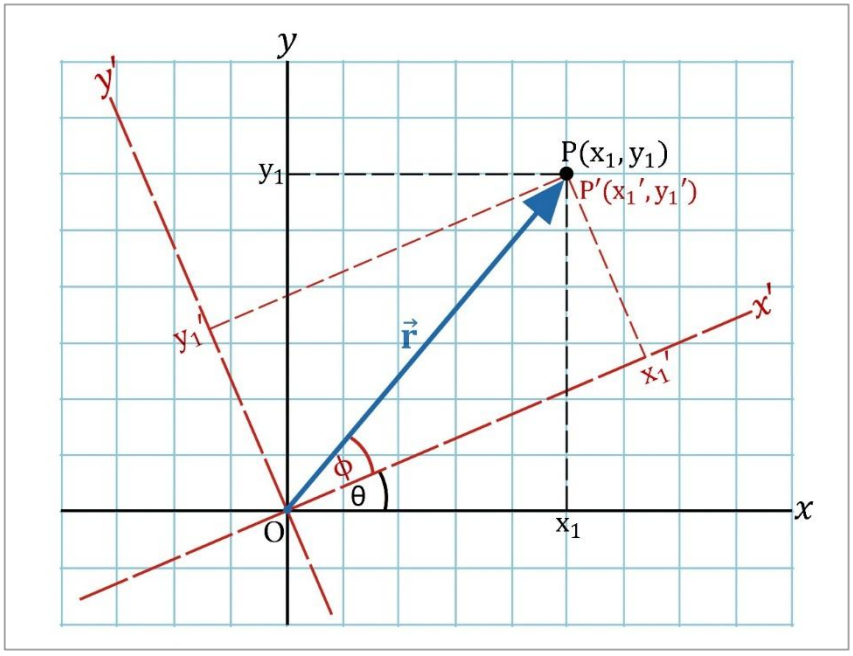

Figura 2. Práctica de vectores – Sistema general.

2.4.1.1. Objetivo de la práctica

Validar experimentalmente los conceptos relacionados con la temática de vectores considerando su importancia dentro del estudio del movimiento y los fenómenos físicos asociados con cantidades vectoriales.

Por medio de la práctica se busca que el estudiante logre determinar y comprobar experimentalmente aspectos puntuales tales como la magnitud y dirección de un

vector resultante, las componentes rectangulares de un vector, la representación de un vector en coordenadas cartesianas y polares, la transformación de un vector de un sistema de coordenadas a otro, las operaciones fundamentales entre vectores, así como también que pueda reconocer las propiedades que distinguen a los vectores de posición y de desplazamiento de un elemento que realice cualquier trayectoria dentro del plano.

2.4.1.2. Definición de la guía de actividades para la práctica

Para la realización de los procedimientos en cada una de las prácticas virtuales se plantea la utilización de una guía de actividades en la que se especifiquen los pasos necesarios para la correcta utilización de la aplicación, independientemente del tema que sea abordado en ellas.

2.4.1.2.1. Estructura general de la guía

Partiendo de esta idea se propone una estructura general para el diseño y la construcción de las diferentes guías de actividades. Tomando como referencia un modelo de guía de laboratorio utilizado en la Universidad autónoma Chapingo (Chapingo, Estado de México) bajo el nombre de “Guía de elaboración de un manual de prácticas de laboratorio, taller o campo: asignaturas teórico - prácticas” [59], se establece una guía de actividades para las prácticas virtuales de física mecánica con los siguientes componentes:

- **Introducción:** En este apartado se da una idea muy general y concisa acerca del tema específico que se va a tratar dentro de la práctica.

- **Objetivo:** Hace referencia al logro general que se quiere alcanzar con la realización de la práctica virtual.

- **Marco teórico:** Expone las definiciones más importantes de los conceptos físicos que serán tratados en cada práctica. Son la base para que el estudiante realice de una manera adecuada las actividades propuestas dentro de la guía.

- **Desarrollo de la práctica:** En esta sección se exponen todos los pasos necesarios para el manejo correcto de la aplicación web. Como resultado de este procedimiento se generan resultados cuantitativos y cualitativos del movimiento que ilustran el experimento físico llevado a cabo en la simulación.

- **Prueba de conocimiento:** Contiene una serie de preguntas relacionadas con el desarrollo del experimento. Es el mecanismo por medio del cual los estudiantes serán evaluados por el docente encargado de realizar la práctica virtual.

2.4.1.2.2. Definición de la guía de vectores

Con base en la estructura general de la guía de actividades (sección 2.4.1.2.1) se plantean dos actividades principales para el desarrollo de la práctica de movimiento vectorial. La primera de ellas es la implementación de un número determinado de vectores en coordenadas polares que representen la trayectoria de un objeto que se mueve dentro de un plano cartesiano, realizada a partir del ingreso de los valores de magnitud y dirección como datos de entrada para cada vector. La dirección de los vectores debe estar especificada por el valor numérico del ángulo (valores entre 0° y 90°) y su orientación respecto a los puntos cardinales que conforman el sistema de referencia cartesiano (Norte, Sur, Este y Oeste).

Al completar el número de vectores ingresados por parte del elemento u objeto encargado de dibujar los vectores, el sistema deberá mostrar la resultante de todos los desplazamientos realizados y los datos numéricos correspondientes a las coordenadas en x y y , la magnitud y dirección (de acuerdo a los puntos cardinales del sistema coordenado) de dicho vector resultante. A partir de estos datos obtenidos el estudiante podrá corroborar matemáticamente los resultados suministrados y determinar otros valores tales como las componentes en x y y de cada vector ingresado.

Por su parte la segunda actividad corresponde a la implementación de vectores en coordenadas cartesianas. De manera similar que en la actividad 1, el estudiante debe realizar el ingreso de un número determinado de vectores que representen la trayectoria de un cuerpo por medio de coordenadas x y y dentro del plano cartesiano. El sistema debe mostrar gráficamente el conjunto de desplazamientos realizados por un objeto dentro del plano cartesiano, el vector resultante de los vectores creados y los resultados numéricos asociados a él: coordenadas en x y y, magnitud y dirección (de acuerdo a los puntos cardinales del sistema coordenado).

Adicionalmente se propone la realización de un experimento más para complementar la segunda actividad, la cual consiste en que a partir de los vectores ingresados en coordenadas cartesianas se realice la rotación de los ejes xy (en sentido antihorario) de acuerdo a un ángulo (entre 0° y 180°) ingresado por el estudiante. De esta manera, al ser rotado el plano cartesiano las coordenadas de los vectores ingresados inicialmente cambian respecto al nuevo sistema de ejes

($x'y'$). La herramienta debe suministrar los valores de las nuevas coordenadas de modo que el estudiante pueda comprobar de forma matemática los resultados obtenidos y además determine la magnitud y dirección de cada vector dentro del nuevo sistema, incluyendo el vector resultante.

2.4.2. Herramientas de desarrollo

Dentro de este apartado se exponen de manera general las funcionalidades que aportan los diferentes elementos que van a ser utilizados dentro del desarrollo de la herramienta de simulación para las prácticas de mecánica. Con base en la elección del lenguaje HTML5 como motor de desarrollo para la aplicación web y considerando la naturaleza del producto que se quiere obtener, se escoge el Framework "Phaser.js" para la implementación de las animaciones de cada uno de los sistemas físicos que componen las seis prácticas que irán contenidas en la aplicación. Algunas de las propiedades y características más importantes de Phaser.js se muestran a continuación:

A. Phaser.js

Como se mencionó en el capítulo 1 (sección 1.4.1), existe una gran variedad de librerías y frameworks especializados en la programación de animaciones y videojuegos con HTML5 y JavaScript, las cuales proveen una buena gama de posibilidades y características útiles para tales propósitos. Una de esas opciones disponibles hoy en día es Phaser, un completo Framework de código abierto que brinda soporte para ciertas tareas dentro del desarrollo de videojuegos tales como implementación de secuencias de animación a base de Sprites, movimiento de personajes, creación de fondos, interacción entre diferentes objetos dentro de los escenarios creados, transiciones, variados tipos de colisiones, manejo de mundos, entre otros [60].

Phaser utiliza Pixi.js, un representador web 2D básico, a través del cual se llevan a cabo tareas de renderización, es decir, procesos de generación de imágenes con base en modelos en tres dimensiones. Phaser también utiliza internamente Canvas en 2D y WebGL, lo que le permite cambiar de uno a otro sin ningún problema, dependiendo de las características del navegador [61]. WebGL es una especificación estándar que permite que el contenido web utilice un API basada en Open ES 2.0 con el objetivo de conseguir representación de gráficos en 3D dentro de un navegador web con formato HTML [62]. El componente Canvas será explicado más adelante.

Aparte de contar con una larga lista de componentes para el desarrollo de videojuegos en la web, Phaser también dispone de algunos objetos y métodos que resultan de mucha utilidad en el desarrollo de los sistemas diseñados para la conformación de la aplicación web. Algunos métodos de Phaser y otros elementos del conjunto HTML5 son tratados a continuación:

- **Sprites:** Un sprite o sprite sheet es una imagen conformada por una serie de cuadros o imágenes más pequeñas que representan una o varias secuencias de animación de un personaje u objeto. Pueden poseer un tamaño grande o pequeño, dependiendo de la cantidad de cuadros que los componen y están ordenados con base en una cuadricula que evita saltos en la secuencia al momento de ser implementados dentro del motor de animación. Esta técnica permite agilizar el proceso de cargar las imágenes de una secuencia ya que en vez de una gran cantidad de cuadros, se crean unos pocos que son reutilizados constantemente, lo que deriva en un bajo consumo de memoria y gran rendimiento por parte del equipo en que se esté realizando el proceso de animación [63].

- **Animation:** Es un método que permite definir las acciones de una animación con base en los cuadros de imágenes contenidos dentro de un sprite. Dicho de otro modo, es el encargado de delimitar los diferentes movimientos llevados a cabo por un personaje u objeto dentro de una animación, de modo que el sistema pueda decidir en qué instante y bajo qué circunstancias una secuencia debe ser empezada o finalizada [64].

- **BitmapData:** Es un objeto que contiene un elemento Canvas con el que se puede realizar cualquier tipo de dibujo o trazado en 2 dimensiones por medio de operaciones de contexto sobre un lienzo dado [65].

- **Interpolación Lineal:** Es un método matemático de Phaser empleado para encontrar la trayectoria lineal que pasa por una serie de puntos dados. Su uso es de gran importancia ya que a partir de este se van a definir los puntos que definirán el recorrido de los sprites en las distintas animaciones. A nivel de código el método se define a través de la línea *linearInterpolation (v,k)*, en donde los parámetros *v* y *k* significan:

    - **v:** Es un parámetro de tipo array con el cual se especifican los valores a interpolar [66].

- **k:** Este parámetro es de tipo numérico y a partir de él se establece el porcentaje de interpolación con valores entre 0 y 1. De esta manera, dependiendo del trayecto formado por los valores de v se retorna un valor por medio de *k*, teniendo en cuenta que 0 es el inicio y 1 es el final de dicho trayecto [67]. El método además retorna un número que depende del porcentaje establecido.

B. Api Canvas

Es un elemento HTML utilizado para realizar diferentes tipos de gráficos o dibujos por medio de scripts (normalmente JavaScript) y para la composición de animaciones de distinta complejidad. El elemento Canvas posee cuatro tipos de contexto [68]:

- **2d:** Guía para la creación de objetos representados en dos dimensiones.

- **WebGL y WebGL2:** Ambos utilizados para la representación de objetos en tres dimensiones, disponibles para navegadores que soporten WebGL en su versión 1 (OpenGL ES 2.0) y 2 (OpenGL ES 3.0) respectivamente.

- **Bitmaprenderer:** Usado para crear mapas de bits especiales para imágenes.

Canvas permite la creación de un cuadro o lienzo sobre el cual se puede graficar, por medio de coordenadas, dentro de un plano similar al cartesiano, con la diferencia de que para este caso el eje y se encuentra invertido, como se muestra en la figura 3:

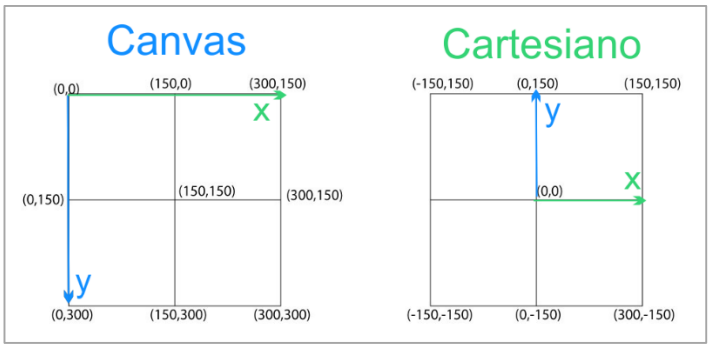

Figura 3. Creación de un lienzo a través de Canvas.

De acuerdo a la figura 3 el punto de origen de las coordenadas inicia en la esquina superior izquierda y el incremento de estas se realiza hacia abajo y a la derecha del lienzo. Su utilización en un documento HTML se realiza a través del elemento <Canvas>, estableciendo un tamaño para el ancho y el alto del cuadro dado en

pixeles (px). De no ser especificados estos parámetros por defecto se creará un lienzo de 300px por 150 px [69].

C. Epoch:

Es una librería de uso general utilizada para la creación de gráficos muy estilizados y de gran funcionamiento. En cuanto a visualización, Epoch se centra en dos aspectos diferentes: gráficos básicos para el soporte de informes históricos y gráficos en tiempo real que muestran la actualización de datos en intervalos definidos. En cuanto a los conjuntos de datos grandes y estáticos la librería ofrece gráficos de área, de barras agrupadas, líneas, circulares y de dispersión. Para funciones especiales como la supervisión en tiempo real Epoch utiliza gráficos basados en Canvas de HTML5, como por ejemplo: gráficos de área, barras, manómetro, mapas de calor y de líneas entre otros [70].

D. Sublime text:

Es un editor de texto muy ligero que aporta un gran número de características de mucha utilidad al momento de programar o de editar código dentro para un buen número de plataformas. El editor contiene una amplia gama de funcionalidades útiles en cuanto a usabilidad y eficiencia, brindando comodidad al proceso completo de edición de texto, convirtiéndolo en una experiencia sencilla y agradable a medida que el usuario va conociendo todas sus funcionalidades. Su licencia de funcionamiento cuesta aproximadamente 50 euros, sin embargo puede ser usado en modo de prueba durante un buen tiempo con todas sus funcionalidades, las cuales abarcan funciones de autocompletado, selecciones múltiples de texto, ediciones múltiples, acceso directo a métodos o funciones, auto-cerrado de etiquetas, entre otras. [71].

E. Medibang Paint pro:

Es un software gratuito usado para la creación de cómics y todo tipo de ilustración digital, el cual contiene una gran variedad de pinceles, herramientas y efectos para dotar de dinamismo a cada uno de los trabajos realizados a partir de él. Dentro del presente trabajo será utilizado para la realización de los diferentes cuadros de imágenes que componen los sprites de cada secuencia de animación. El software está disponible para PC, Mac y dispositivos móviles [72].

Nota: Los elementos anteriormente estudiados conforman la base para llevar a cabo el desarrollo de todos los sistemas que van a formar parte de la herramienta de simulación final, por tal motivo a partir de la segunda práctica y hasta el final se omite el apartado correspondiente al estudio de las herramientas de desarrollo.

2.4.3. Diseño de la aplicación

La práctica de vectores está basada en las dos actividades definidas inicialmente (actividad 1 y actividad 2), por lo tanto, el diseño correspondiente de la herramienta de simulación viene dado de la siguiente forma:

2.4.3.1. Actividad 1

Con base en la definición de la guía de actividades se diseña el correspondiente algoritmo que represente la secuencia de instrucciones necesaria para la realización de la actividad 1 dentro de la guía de vectores, como se muestra en la figura 4. Los pasos contenidos dentro del diseño son los siguientes:

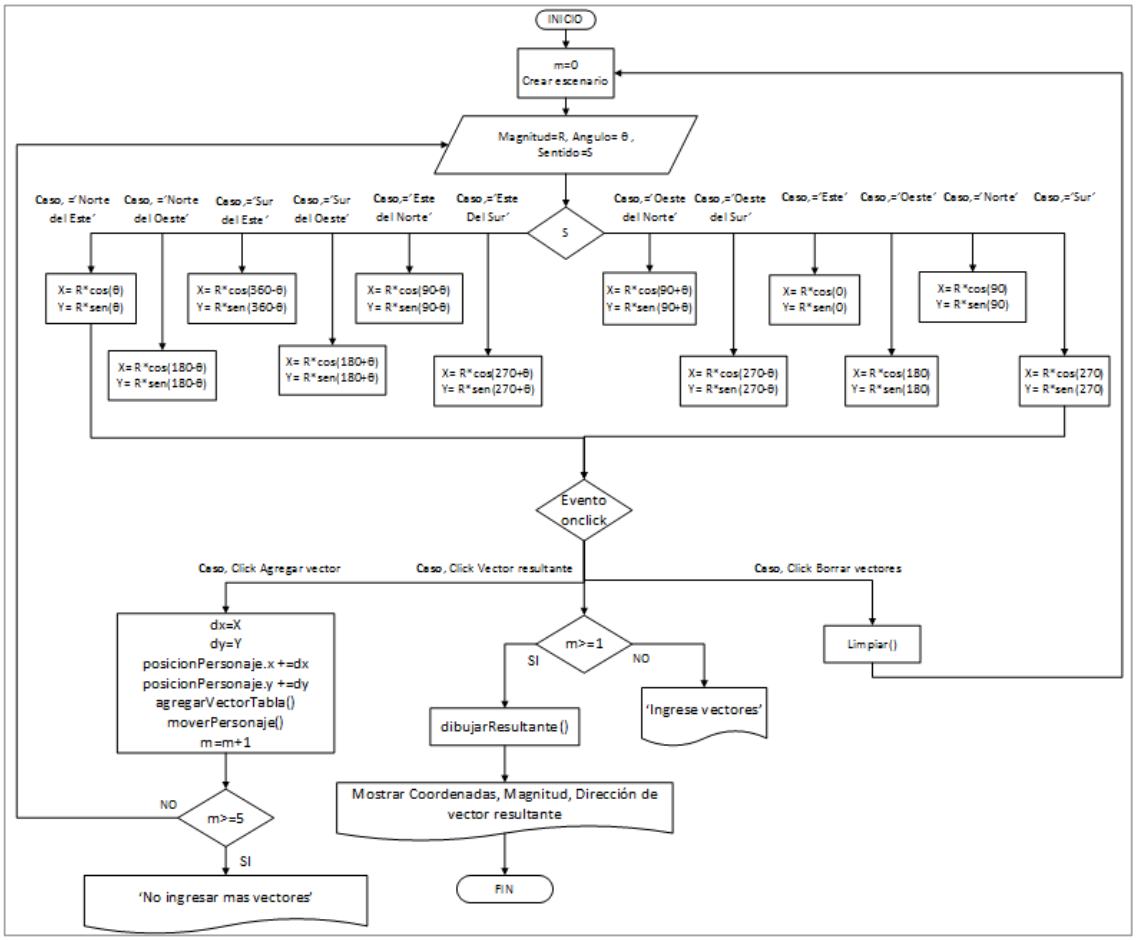

Figura 4. Algoritmo de la actividad 1 de la práctica de vectores.

En la estructura del algoritmo se observa que inicialmente debe haber un proceso en el cual se declara una variable $m = 0$, encargada de controlar el número de vectores que se ingresan, de modo que puedan ser registrados en una posición determinada dentro de una tabla. Este proceso además debe contar con una función que sea la encargada de crear el escenario de la simulación, compuesto por el fondo, un plano cartesiano y un personaje animado que se irá desplazando a lo largo de este plano conformando gráficamente los vectores ingresados.

Posteriormente se reciben los datos de entrada del sistema, los cuales corresponden a la magnitud (R), dirección (θ) y orientación (S) de cada uno de los vectores que se van consignando. En cuanto al dato de orientación, se van a considerar doce posibles casos para ingresar: Norte, Sur, Este, Oeste, Norte del este, Norte del Oeste, Sur del este, Sur del Oeste, Este del Norte, este del Sur, Oeste del Norte y oeste del Sur. De acuerdo a la opción escogida para la variable S se establecen las expresiones matemáticas para cada caso, de modo que se puedan representar los vectores ingresados en función de sus posiciones x y y. Este proceso es necesario ya que para realizar la animación es necesario conocer la posición de cada vector, de modo que a partir de esta información el sistema internamente realice la conversión de kilómetros a pixeles adaptándolos al escenario de simulación.

Cuando internamente se obtenga la posición de cada vector en términos de x y y todo el comportamiento de la aplicación va a depender de los botones que se creen para realizar los diferentes procedimientos que requiera la simulación. De acuerdo a la mecánica del sistema planteado se hace necesario la utilización de tres botones para llevar a cabo las acciones correspondientes a la adición de vectores (“agregar Vector”), despliegue del vector resultante (“vector resultante”) y la eliminación de los vectores añadidos (“borrar vectores”). Dentro del algoritmo se observa que estas acciones son evaluadas por medio de un condicional en el cual se determina si el usuario realiza click sobre alguno de los botones (a través del evento *onclick*). Los tres casos posibles del evento *onclick* se describen a continuación:

A. Caso 1: Click en botón “agregar vector”

En este punto se debe definir la lógica que permita llevar a cabo la animación del sistema planteado, para ello se cuenta con un proceso que contiene inicialmente las variables dx y dy, las cuales toman el valor de posición en pixeles calculadas a partir de los doce casos nombrados anteriormente. Estas variables son necesarias ya que la posición del personaje que forma los vectores va a depender de ellas.

Otras de las variables contenidas dentro del proceso se definen como: $posicionPersonaje.x \mathrel{+}= dx$ y $posicionPersonaje.y \mathrel{+}= dy$, utilizadas en la suma de los valores de posición en x y y a medida que se ingresa un nuevo vector.

A partir de los valores obtenidos dentro de estas variables se conformará la trayectoria seguida por el personaje de la siguiente forma: inicialmente se debe ubicar al personaje en la posición (0,0) del plano cartesiano y cuando sea ingresado un vector por medio de la posición (x_1, y_1) este recorrerá el trayecto desde el origen de coordenadas (0,0) hasta dicho punto (x_1, y_1). Si un nuevo vector de coordenadas (x_2, y_2) es introducido, el personaje partirá ahora desde este punto hasta llegar a una nueva posición dada por $(x_1 + x_2, y_1 + y_2)$, y así sucesivamente hasta alcanzar la posición final, la cual estará definida por la suma de las posiciones en x y y de cada uno de los segmentos orientados.

Así mismo, cuando es presionado este botón son llamadas internamente un par de funciones más, denominadas **agregarVectorTabla ()** y **MoverPersonaje ()**, con las cuales serán registrados los vectores ingresados en la respectiva posición de la tabla (dependiendo del valor de la variable m) y se definirán los puntos que deberá recorrer el personaje respectivamente. En esta última función se activarán también todas las secuencias de animación definidas previamente. Finalmente, cada vez que sea presionado este botón la variable m debe ir aumentando en una unidad y mientras el valor sea menor o igual que 5 se dará la opción de seguir ingresando otros vectores. Cuando el límite impuesto sea alcanzado se debe indicar a través de un mensaje de alerta que ya no es posible seguir añadiendo más de estos.

B. Caso 2: Click en el botón "vector resultante"

Este botón es el encargado de graficar la resultante de todos los vectores ingresados. Si $m \geq 1$ (más de un vector ingresado) se llama a la función **dibujarResultante ()**, la cual se encarga de trazar gráficamente el segmento orientado que une el punto de origen del plano cartesiano con el punto que representa la posición del último vector añadido. También, si la condición es verdadera, se muestran los respectivos datos de salida del sistema, los cuales corresponden a la representación en coordenadas cartesianas x y y, magnitud y dirección del vector resultante obtenido. En caso de que no se cumpla la condición ($m \geq 1$) el sistema debe desplegar un mensaje que indique al usuario la necesidad de ingresar vectores.

C. Caso 3: Click en el botón “borrar vectores”

Cuando es presionado este botón automáticamente se borran todos los vectores agregados por medio de una función denominada **limpiar ().**

2.4.3.1.1. Ecuaciones de movimiento

Las ecuaciones que definen los valores numéricos del vector resultante vienen dadas de la siguiente manera:

Sean (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) y (x_5, y_5) las coordenadas de los vectores de posición $\overrightarrow{r_1}$, $\overrightarrow{r_2}$, $\overrightarrow{r_3}$, $\overrightarrow{r_4}$ y $\overrightarrow{r_5}$ respectivamente, ingresados dentro del plano, las coordenadas cartesianas del vector de desplazamiento resultante $\vec{R}$, representadas por R_x y R_y vienen dadas como:

$$R_x = x_1 + x_2 + x_3 + x_4 + x_5 \quad \textbf{(1)}$$

$$R_y = y_1 + y_2 + y_3 + y_4 + y_5 \quad \textbf{(2)}$$

Magnitud del vector resultante (R):

$$R = \sqrt{{R_x}^2 + {R_y}^2} \quad \textbf{(3)}$$

Dirección del vector resultante (θ_R):

$$\theta_R = \tan^{-1}\left(\frac{R_y}{R_x}\right) \quad \textbf{(4)}$$

2.4.3.2. Actividad 2

El algoritmo desarrollado para la implementación de la actividad 2, mostrado en la figura 5 es similar al de la actividad 1, con la variación de que en este caso los datos que se deben ingresar corresponden a la posición en x y y de cada vector dado, de esta manera se omite el paso de la conversión de coordenadas polares a cartesianas.

También, el algoritmo cuenta con un botón adicional que se utiliza para la rotación del plano cartesiano donde se han ingresado los vectores. Para ello el sistema debe

disponer de una variable de entrada denominada ángulo de rotación del plano (θ_r) para indicar el valor del ángulo que se desean rotar los ejes coordenados (xy). Se debe ingresar por lo menos un vector para realizar la rotación, de esta manera, si se cumple que $m \geq 1$ se llaman las funciones **Rotar Eje ()** y **VectoresRotadoTabla ()** que son las encargadas de dibujar los ejes coordenados en la nueva posición y de mostrar a través de una tabla las nuevas coordenadas de los vectores dentro del plano rotado respectivamente. Por su parte, si no se cumple la condición se debe mostrar un mensaje indicando que se requiere el ingreso de un vector.

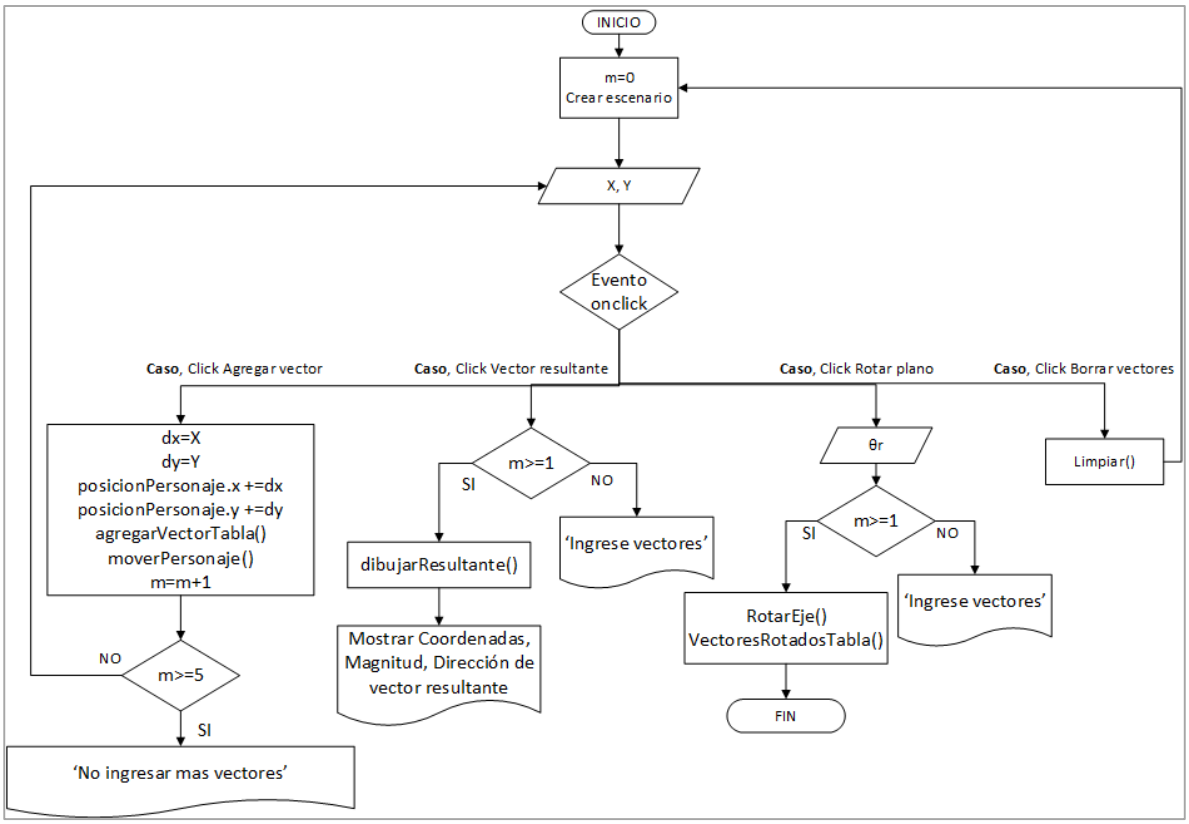

Figura 5. Algoritmo de la actividad 2 - Práctica de vectores.

2.4.3.2.1. Ecuaciones de movimiento

De acuerdo a la figura 2, los valores de las nuevas coordenadas de un vector respecto a un plano cartesiano que ha sido rotado θ grados y que están definidas como (x', y') vienen definidas a través de las siguientes expresiones:

$$x = x'\cos\theta - y'\sin\theta \quad \textbf{(5)}$$

$$y = x'\sin\theta + y'\cos\theta \quad \textbf{(6)}$$

Nota: El desarrollo completo de las expresiones físicas de la práctica de movimiento vectorial se encuentra disponible en el numeral 1 del anexo A (ecuaciones de movimiento de los sistemas físicos diseñados).

2.4.4. Proceso de codificación: práctica de vectores

La implementación de la aplicación correspondiente a la práctica de vectores se lleva a cabo por medio de las herramientas de desarrollo definidas en la sección 2.4. Una de esas herramientas corresponde al framework Phaser.js, elemento que contiene una estructura completa ya definida dentro de la cual debe ser adherida la lógica de simulación del sistema que se desea construir. A continuación se detalla el procedimiento general del desarrollo de la aplicación por medio del framework Phaser.js que será utilizado en la implementación de todas las prácticas definidas:

2.4.4.1. Procedimiento general del desarrollo en phaser.js

En términos generales el procedimiento empieza con la creación de un archivo HTML en cuyo contenido debe ir especificada la etiqueta con su respectivo identificador, en la cual se llevan a cabo las diferentes animaciones. La sintaxis que sigue dicho procedimiento es la siguiente:

* *

Posteriormente se crea un archivo JavaScript (.js) dentro del cual se estructura la parte funcional de la simulación. En él se define inicialmente el ancho y el alto de dicha simulación a través de una variable encargada de hacer la relación de pixeles a metros (o Kilómetros), como se explicará en un punto posterior. Una vez definidas estas dos variables se sigue la estructura de desarrollo de Phaser dentro del archivo .js por medio de los pasos mostrados a continuación:

1. Se agrega la siguiente línea de código:

game = new Phaser.Game (ancho, alto, Phaser.CANVAS, 'juego');

En donde se definen algunos aspectos importantes tales como:

a) El tamaño que ocupara el de la simulación, indicados en los dos primeros parámetros.

b) El modo de ejecución del motor de Phaser y el manejo de gráficas definido en el tercer parámetro, el cual puede tomar las opciones Phaser.AUTO, Phaser.CANVAS y Phaser.WEBGL, como se indica en la tabla 6 [73]:

Tabla 6. Propiedades para el manejo visual de Phaser.

Nombre	Descripción
Phaser.AUTO	Otorga al motor el mejor modo de ejecución y le permite definir el componente para mostrar los gráficos. Esto dependerá de la plataforma en la que se esté ejecutado el videojuego.
Phaser.CANVAS	Representa la versión básica del manejo de gráficos.
Phaser.WEBGL	Utiliza exclusivamente el componente WebGL para el manejo de gráficos. Los dispositivos que no cuenten con el soporte para esta tecnología no podrán visualizar el juego.

c) El nombre del identificador de la etiqueta a la cual será enviada la simulación, cuyo nombre para este caso es “juego”.

2. Se definen las propiedades iniciales de la simulación a través de una función dada por la siguiente expresión:

var LogicaJuego = function () {}

Al ser creada la función se deben establecer los valores iniciales de las variables requeridas para la lógica de la simulación. Dado que Phaser es un Framework diseñado para la realización de videojuegos, el cual está basado en estados (cargar archivos, menú del juego, jugar en diferentes niveles, entre otros), se requiere para este caso crear un estado mediante game.state.add (), dentro del cual irán cargadas todas las funciones de la simulación. De manera general, dicho estado viene definido como:

LogicaJuego.prototype = {

Init: function (){},

Preload: function (){},

Créate: function (){},

Update: function (){},

}

game.state.add('Game', LogicaJuego, true)

Donde el primer parámetro de la línea final del código corresponde al nombre que tendrá el estado en caso de que se necesite ser usado posteriormente. El segundo se refiere al nombre de la variable que contiene el objeto que se va a usar como estado, el cual se ha llamado 'Logicajuego'. Por último el tercer parámetro indica si el estado inicia apenas se carga la página. En caso de que este parámetro no se defina, la forma de iniciar un estado es de la siguiente manera:

game.state.start ('Game');

Dentro del código utilizado para definir el estado se observa una serie de funciones definidas por Phaser resumidas dentro de la tabla 7 [61]:

Tabla 7. Funciones definidas en Phaser.

Función	Características
Init	Empleada para definir los parámetros iniciales de la simulación.
Preload	Función que permite cargar las imágenes que van a ser utilizadas en las secuencias de animación dentro de la simulación.
Create	Función utilizada para la creación de los fondos, los sprites, las animaciones, entre otros. A través de ella son llamadas las otras funciones que están asociadas con los botones creados.
Update	Es una función que tiene la propiedad de actualizarse constantemente. En ella se especifica la lógica para realizar las respectivas animaciones.

Como se mencionó en la definición de la estructura general del desarrollo con Phaser, es necesario contar con una variable para realizar la equivalencia entre pixeles y Kilómetros dentro de la animación, esto con el fin de establecer una relación entre las dimensiones digitales y las dimensiones físicas de los objetos que componen el sistema que se quiere representar. La variable en cuestión ha sido denominada *diezU* y está definida de la siguiente manera:

$$diezU = \frac{(ancho - 100)}{14} \quad \textbf{(7)}$$

Donde el término $(ancho - 100)$ se emplea para definir el ancho que se va a utilizar para construir el plano cartesiano de ejes coordenados. El número 100 indica que a cada lado del eje horizontal se va a dejar a dejar un espacio de aproximadamente 50 pixeles. Este valor de divide por un valor igual a 14, lo que significa que dentro del eje x se van a marcar las posiciones que van desde -70 hasta 70 en pasos de 10 unidades. Por su parte el ancho del plano y el alto de la simulación equivalen respectivamente a:

$$ancho = 1366\,pixeles * 0.6 \quad \textbf{(8)}$$

$$alto = 768\,pixeles \quad \textbf{(9)}$$

De esta manera, la variable *diezU* equivale a:

$$diezU = \frac{(1366 * 0.6 - 100)}{14} = 51.4\,pixeles \quad \textbf{(10)}$$

Lo que significa que 10 Kilómetros equivalen a 51.4 pixeles, valor que ha sido escogido por conveniencia.

Los valores de ancho y alto se tomaron con base a las dimensiones del ordenador sobre el cual se lleva a cabo la implementación, es decir sobre un equipo portátil con resolución de pantalla de 1366x768 pixeles. En caso de que se requiera realizar un diseño adaptativo, es necesario dejar el ancho de la simulación en función del tamaño de la pantalla, para ello se utilizan algunas propiedades disponibles tales como screen.width o window.innerWidth. Del mismo modo se debe hacer para el alto, en caso de que esto suceda la relación pixeles – metros varía dependiendo del valor ancho de la pantalla.

Después de haber definido los parámetros básicos de simulación para la práctica de vectores se procede a la definición de algunas funciones importantes que permiten la implementación de los procesos de animación del sistema diseñado. Dichas funciones son las siguientes:

- **Función Preload ():**

Permite cargar las imágenes y los sprites que se van a incluir dentro del escenario de la animación. Las imágenes que acompañan la interfaz de la práctica de vectores corresponden a una rosa de los vientos (archivo rosaw.png) que indica las posibles orientaciones que pueden llegar a tomar los vectores dibujados en la simulación y la imagen de una flecha o punta (roja.png y azul.png) para referenciar la dirección de los vectores ingresados y del vector resultante respecto al plano cartesiano. Dentro de la aplicación se hace uso de un sprite conformado por una serie de imágenes que ilustran una secuencia animada para la creación de todos los vectores ingresados por el usuario (tanto la implementación de la secuencia de animación como el respectivo sprite son tratados más adelante), el procedimiento para cargar las imágenes se realiza a través de la siguiente línea de código dentro del archivo .js:

game.load.spritesheet ('personaje', 'Actividad1/imágenes/personaje.png', 32, 48)

Donde el primer parámetro de la línea corresponde al nombre que se le ha dado al sprite que contiene la secuencia de movimientos que realiza el personaje para crear cada uno de los vectores ingresados y el segundo hace referencia a la ubicación de dicho archivo. Similarmente el tercer y cuarto parámetro indica el tamaño del archivo correspondiente al sprite. La figura 6 muestra la manera en que queda definida la función **Preload ()** dentro del código de la simulación.

```
preload: function () {

    game.load.spritesheet('personaje','Actividad1/imagenes/personaje.png', 32, 48)
    game.load.spritesheet('puntosc','Actividad1/imagenes/rosaw.png')
    game.load.spritesheet('trian','Actividad1/imagenes/roja.png')
    game.load.spritesheet('trianr','Actividad1/imagenes/azul.png')

},
```

Figura 6. Definición de la función Preload ().

- **Función Create ():**

Es la función encargada de realizar algunos procesos dentro de la simulación, como por ejemplo la implementación del color de fondo del área donde se lleva a cabo la animación, el llamado a las funciones que dibujan el sistema de ejes coordenados, la definición del orden de las imágenes en la secuencia dentro del sprite, entre otras. Inicialmente se emplea el objeto BitmapData, el cual contiene un elemento Canvas que permite la creación de fondos, figuras y cuadros de texto dentro de un escenario

de simulación. En la figura 7 se observa la forma en que se agrega el objeto al código JavaScript:

```
bmd = this.add.bitmapData(this.game.width, this.game.height)
bmd.addToWorld()
ctx=bmd.context;
```

Figura 7. Creación del objeto BitmapData.

La primera línea indica el tamaño del lienzo que se va a utilizar, correspondiente a todo el div que contiene la simulación. En la segunda línea se agrega al mundo el BitmapData creado, seguido de la creación del contexto 2D para Canvas, como se indica en la tercera línea de código. A partir de esto es posible acceder a todos los métodos de la API de Canvas, como por ejemplo **beginPath ()**, **closePath ()**, **stroke ()**, **fill ()**, **createLinearGradient (x_1,y_1, x_2, y_2)**, entre otros. Por su parte la figura 8 muestra la estructura de los pasos requeridos para la creación del fondo donde se realiza la simulación:

```
ctx.beginPath();
var grd = ctx.createLinearGradient(0,0,0,this.game.height);
grd.addColorStop(0.5, '#22B197');
grd.addColorStop(1, '#24F3CD');
ctx.fillStyle= grd;
ctx.fillRect(0,0, this.game.width, this.game.height);
```

Figura 8. Creación del objeto BitmapData.

En dicha estructura se llevan a cabo los siguientes procedimientos:

a) Inicialmente se emplea el método **beginPath ()** para indicar que se va a iniciar un nuevo trazado, como se ve en la línea 1. Este debe ser colocado cada vez que se desee realizar una nueva figura a través de Canvas.

b) Seguidamente en la línea 2 se crea el objeto gradiente, en el cual se especifica la inclinación de la transición o el degradado de los colores mezclados por medio de las variables (x_1,y_1, x_2, y_2).

c) Posteriormente con el método **addColorStop ()** se determinan los colores y la posición en el gradiente a través de dos parámetros (líneas 3 y 4 del código). El primero de ellos consta de un valor numérico (entre 0 y 1) en el que se indica el comienzo y el final de los colores que conforman el gradiente.

El segundo parámetro indica el color que se va a agregar para la conformación de este gradiente.

d) Con la propiedad fillStyle se define el color que se va a aplicar a una determinada figura, que en este caso consiste en el gradiente anteriormente creado (línea 5).

e) Finalmente, dado que se busca la creación del fondo para la animación, es necesario crear un rectángulo con las dimensiones del de la simulación y rellenarlo posteriormente. Para ello se emplea el método **fillRect (x,y, ancho, alto)**, cuyos parámetros x y y indican el punto de inicio del trazado. Los dos parámetros restantes hacen alusión al ancho y al alto del rectángulo respectivamente. De esta manera queda definido el fondo para la simulación.

Después de que se ha creado el fondo es el momento de agregar las imágenes y el sprite que contiene la secuencia de figuras para animar la construcción de cada uno de los vectores que se vayan ingresando. La figura 9 muestra el sprite que se ha utilizado dentro de la simulación, en él se observan las diferentes posiciones que adopta el personaje cuando recorre el plano implementando gráficamente cada vector:

Figura 9. Sprite utilizado para la creación de los vectores.

Cabe aclarar que dentro del presente trabajo se realizaron por cuenta propia la gran mayoría de las imágenes y de sprites utilizados en la implementación de las diferentes simulaciones, sin embargo, el sprite de la figura 9 fue obtenido de una de las simulaciones gratuitas disponibles en la página oficial de Phaser [74].

Este sprite fue agregado a la simulación de la siguiente manera (ver figura 10):

```
this.personaje = game.add.sprite(this.centro.x, this.centro.y, 'personaje')
this.personaje.anchor.set(0.5, 1)

this.personaje.animations.add('derecha', [5, 6, 7, 8], 6, true)
this.personaje.animations.add('centro', [4], 6, true)
this.personaje.animations.add('izquierda', [0, 1, 2, 3], 6, true)
```

Figura 10. Código para agregar el sprite.

En la primera línea de código se indica la posición inicial del sprite y el nombre asignado dentro de la función **Preload ()**. Por su parte, en la segunda línea se establece el punto de referencia del sprite a través de la propiedad anchor.set (0.5, 1), en este caso la mitad inferior de la imagen ubicada en la posición 4 dentro de dicho sprite, la cual será puesta en el punto central del rectángulo creado, coincidiendo con el origen del sistema coordenado implementado al interior de dicho rectángulo.

Las líneas 3, 4 y 5 contienen el proceso de animación del personaje creado a partir de las nueve imágenes contenidas en el sprite. Para ello se han definido tres animaciones denominadas derecha, centro e izquierda respectivamente, donde en cada una se especifican las imágenes o los frames utilizados por medio de su posición dentro del sprite, seguida del valor de velocidad asignado para el cambio de dichas imágenes.

Además de la creación del fondo y del manejo de las imágenes incluidas en la simulación, en la función **Create ()** se definen también las funciones asociadas a un gran número de eventos, como por ejemplo el de hacer click sobre los botones de la interfaz. Dicha funcionalidad, llevada a cabo a través del evento denominado onclick, permite la implementación de tres tipos de botones dentro de la práctica de vectores denominados agregar, resultante y limpiar.

Por ejemplo, en la figura 11 se observa la manera de definir la función que permite realizar un click sobre el botón resultante, la cual se basa en determinar si la variable $this.enMovimiento = false$, es decir, si la simulación ha finalizado y el personaje en movimiento ha completado su recorrido, se llama a la función **DibujarResultante ()**, la cual se encarga de trazar gráficamente el vector resultante o vector desplazamiento del sistema. De manera similar se definen las funciones que involucran los botones restantes.

```
document.getElementById('resultante').onclick = function (ev) {
  if (!this.enMovimiento) {
      this.dibujarResultante()
  }
}.bind(this)
```

Figura 11. Función para el evento onclick.

Sin embargo las funciones más importantes son las que están asociadas con el botón agregar**,** ya que cada vez que este es presionado se agrega un nuevo vector a la simulación y se consigna dentro de una tabla de vectores ingresados por el usuario. Por otro lado, la práctica contiene dos actividades orientadas a la

representación de vectores tanto en coordenadas polares (actividad 1) como cartesianas (actividad 2). La actividad 1 brinda al usuario la opción de ingresar un vector por medio de su magnitud y dirección respecto a alguno de los ejes coordenados, mientras que la actividad 2 permite el ingreso de cada vector a través de sus coordenadas dentro del plano xy.

En la actividad 1 es necesario obtener internamente los vectores de posición en términos de las coordenadas x y y para cada pareja de valores de magnitud y dirección consignadas en la tabla. Para ello se deben considerar doce casos de acuerdo a la dirección de los vectores ingresados (norte, sur, este, oeste, norte del este, sur del este, etc.), los cuales deben ser definidos antes de llamar a las funciones que permiten ingresar dichos vectores y que generan los movimientos del personaje.

Tales funciones fueron llamadas **AgregarVectorTabla (m)** y **moverPersonaje (dx, dy)** respectivamente, las cuales dependen de los parámetros dx y dy, correspondientes a la posición (x,y) del vector dada en pixeles, hallada después de realizar cada uno de los doce casos mencionados, y un parámetro m que activa un contador cada vez que el botón agregar es presionado, siempre y cuando el número de vectores ingresados no haya sobrepasado el límite establecido.

En la función **MoverPersonaje ()** se define la trayectoria que debe seguir el personaje al momento de construir los vectores ingresados por el usuario. La figura 12 muestra en detalle el código relacionado con esta función, en la cual se empleó el método de interpolación lineal para el trazado de dicha trayectoria:

```
moverPersonaje: function (dx, dy) {

    this.posicionPersonaje.x += dx
    this.posicionPersonaje.y += dy

    this.camino = []
    var x = 1 / this.pasos

for (var i = 0; i <= 1; i += x) {
  var px = this.math.linearInterpolation([this.posicionPersonaje.x - dx, this.posicionPersonaje.x], i)
  var py = this.math.linearInterpolation([this.posicionPersonaje.y - dy, this.posicionPersonaje.y], i)

  this.camino.push( { x: px, y: py })
    }
    this.enMovimiento = true
    if (dx > 0)
        this.personaje.animations.play('derecha')
    else
        this.personaje.animations.play('izquierda')

    this.paso = 0
},
```

Figura 12. Función encargada de determinar la trayectoria del personaje.

La estructura del código muestra que la función recibe dos parámetros que corresponden a la posición (x,y) de cada vector ingresado expresada en pixeles, y dado que Inicialmente el personaje se encuentra en el centro del plano cartesiano, es decir en la posición (0,0), su ubicación dentro de la simulación corresponde a la pareja de coordenadas (this.game.width/2, this.game.height/2). Las variables $this.posicionPersonaje.x$ y $this.posicionPersonaje.y$ van cambiando a medida que se ingresa un nuevo vector, de modo que la nueva posición corresponde a la suma de las posiciones de los vectores anteriormente ingresados.

Con base en esto, si por ejemplo se ingresa un primer vector con una posición (x_1, y_1) el personaje realizará su recorrido desde el origen de coordenadas (0,0) hasta el punto representado por (x_1, y_1). Si es ingresado un nuevo vector mediante la posición (x_2, y_2), el personaje recorre desde el punto (x_1, y_1) hasta la posición $(x_1 + x_2, y_1 + y_2)$, y así sucesivamente hasta que se complete el ingreso de los cinco vectores requeridos para la simulación. Dicho proceso de la suma de posiciones se observa en la segunda y tercera línea de código de la figura 13.

Para la conformación de los vectores se recurre al método de interpolación lineal, con el cual se determinan los puntos que hacen parte de la trayectoria seguida por el personaje. Para ello es necesario conocer previamente la ubicación de algunos de estos puntos como referencia para realizar el trazado del trayecto, sin embargo para este caso solo basta con conocer las posiciones inicial y final del vector en términos de x y y. A través de un ciclo for, el software internamente va guardando dentro de un arreglo los valores de x y y ingresados por el usuario.

Por medio de una variable definida como this.enMovimiento se indica al sistema el instante en el que puede iniciar el movimiento del personaje. Este proceso se lleva a cabo dentro de una función que se ejecuta constantemente denominada **Update ()**, siempre y cuando dicha variable tenga como valor *true*. Por su parte, la simulación se activa por medio de $this.personaje.animations.play$ (‘ ’) con base en la dirección en la que se vaya a realizar el movimiento.

- **Función Update ():**

A través de esta función de Phaser se implementa el proceso de animación del movimiento del personaje con base en la secuencia de imágenes contenidas en el sprite, como se aprecia en la figura 13.

```
update: function () {

    var teta= parseFloat(document.getElementById("teta").value*Math.PI/180)
    if (this.enMovimiento) {
        var px = this.camino[this.paso].x
        var py = this.camino[this.paso].y
        this.personaje.x = px
        this.personaje.y = py
        this.bmd1.rect(px, py, 2, 2, 'rgba(255, 255, 255, 1)'); //Crea el vector
        if (this.paso < this.pasos - 1) {
            this.paso++
        } else {


            this.enMovimiento = false
            this.personaje.animations.stop()
            this.bmd1.circle(px, py, 4, 'rgba(255, 0, 0, 1)')
            this.personaje.animations.play('centro')
            this.personaje.animations.stop()
```

Figura 13. Definición de la función Update ().

De acuerdo a la figura 13, se requiere también la creación de unas variables denominadas px y py las cuales van tomando los valores de los puntos de la trayectoria calculados en la función **moverPersonaje ()**. Estas variables son asignadas a la posición del personaje a través de $this.personaje.x = px$ y $this.personaje.y = py$. De acuerdo al valor de un contador, el personaje recorre cada uno de los puntos calculados con el objetivo de ir conformando gráficamente, por medio de pequeños rectángulos de tamaño 2 x 2 pixeles, los vectores que se van ingresando. Todo esto se realiza por medio del siguiente método:

this.bmd1.rect (px, py, 2, 2, 'rgba (255, 255, 255, 1)');

En donde *this.bmd1* consiste en otro objeto del BitmapData creado para que el trazo quede posicionado por encima del sprite. Cuando el valor del contador alcance el límite de puntos calculados para la trayectoria se debe indicar en la función **MoverPersonaje ()** que el proceso ha terminado, asignando el valor *false* a la variable $this.enMovimiento$. También se debe detener la animación mediante la propiedad stop.

2.5. PRÁCTICA 2 - MOVIMIENTO EN DOS DIMENSIONES

2.5.1. Diseño del sistema

La figura 14 muestra el esquema general del sistema diseñado para la práctica de movimiento en dos dimensiones. Dicho sistema se compone de un cuerpo esférico de masa m (pelota) y dos rampas inclinadas colocadas de manera contigua, las cuales poseen ángulos de elevación variables (θ_1 y θ_2 respectivamente), igual longitud en su base inferior (40 metros) y sus superficies no presentan fricción.

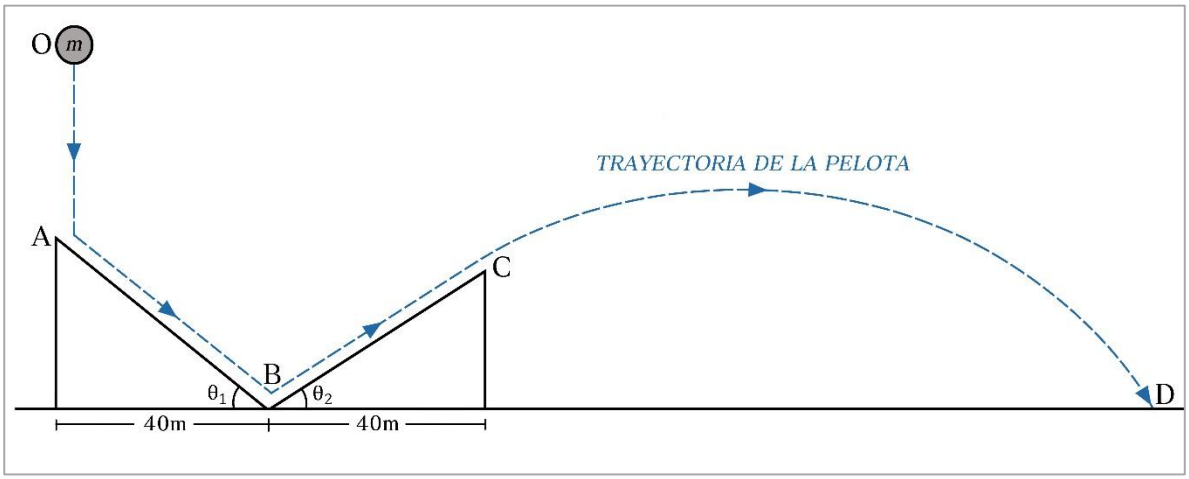

Figura 14. Práctica de movimiento bidimensional – Sistema general.

El funcionamiento del sistema es el siguiente: En un instante inicial se ubica la pelota a una determinada altura por encima de la rampa 1 y se deja caer, iniciando un movimiento de caída libre hasta llegar a la punta de dicho plano inclinado. Dentro de este movimiento vertical solo se considera la aceleración del cuerpo debido a la gravedad y se desprecian los efectos causados por la fricción del aire. Al llegar a la primera rampa la pelota realiza otro tipo de movimiento, esta vez sobre un plano inclinado descendente, partiendo con la misma velocidad con la que finalizó su movimiento de caída libre.

En un instante posterior, luego de atravesar la longitud de dicha rampa, el cuerpo se ubica en el punto de inicio del segundo plano inclinado. De forma similar se considera que la pelota parte con una velocidad inicial igual a la velocidad con la que finalizó su desplazamiento en la rampa 1. Si este valor es suficiente la pelota podrá atravesar totalmente la rampa 2 y realizar un movimiento de tipo parabólico, dentro del cual solo se tiene en cuenta la aceleración producida por la gravedad y se desprecian los efectos resistivos del aire. En resumen, el movimiento total de la pelota se realiza a través de cuatro tramos especificados en la tabla 8:

Tabla 8. Movimiento bidimensional – Tramos del sistema.

	Tipo de Movimiento	Tramo
1	Movimiento de caída libre	O-A
2	Movimiento sobre un plano inclinado descendente	A-B
3	Movimiento sobre un plano inclinado ascendente	B-C
4	Movimiento parabólico	C-D

2.5.1.1. Objetivo de la práctica de movimiento bidimensional:

Fortalecer los conceptos de cinemática gracias al estudio del movimiento de un cuerpo dentro de un sistema definido por secciones, el cual permite la realización de actividades independientes que faciliten la observación de las características propias de cada uno de los subsistemas que lo componen.

A través de la práctica el estudiante podrá comprobar matemáticamente los valores de aceleración, velocidad y posición del cuerpo en cualquier punto del sistema, con base en unas condiciones iniciales de altura de la pelota y ángulos de elevación de cada una de las rampas.

2.5.2. Diseño de la aplicación

La figura 15 muestra el esquema del algoritmo diseñado para la práctica de movimiento en dos dimensiones. La secuencia de pasos muestra que Inicialmente el sistema recibe los datos de entrada correspondientes a la altura de ubicación del objeto (h), los ángulos de elevación de las rampas 1 (θ_1) y 2 (θ_2) y la velocidad inicial con que la pelota parte dentro de la rampa ascendente (v_0), considerada solamente dentro de la simulación de tramo B-C-D. Posteriormente se tiene un proceso en el que se define la variable $balónMovimiento = false$, encargada de indicar si la pelota que recorre el trayecto definido está en movimiento o no.

Dentro de este proceso también son llamadas las funciones encargadas de inicializar las gráficas (trazado de los ejes coordenadas de cada gráfica) y de crear el escenario de simulación, compuesto por las dos rampas especificadas en la descripción del sistema, un balón que recorre los trayectos definidos y un elemento que ubique dicho balón a una altura determinada del suelo. Se define además una variable k inicializada en cero que indica si hay cambios o no en la entrada de datos, esto con el fin de saber si se requiere la actualización del escenario para una nueva simulación.

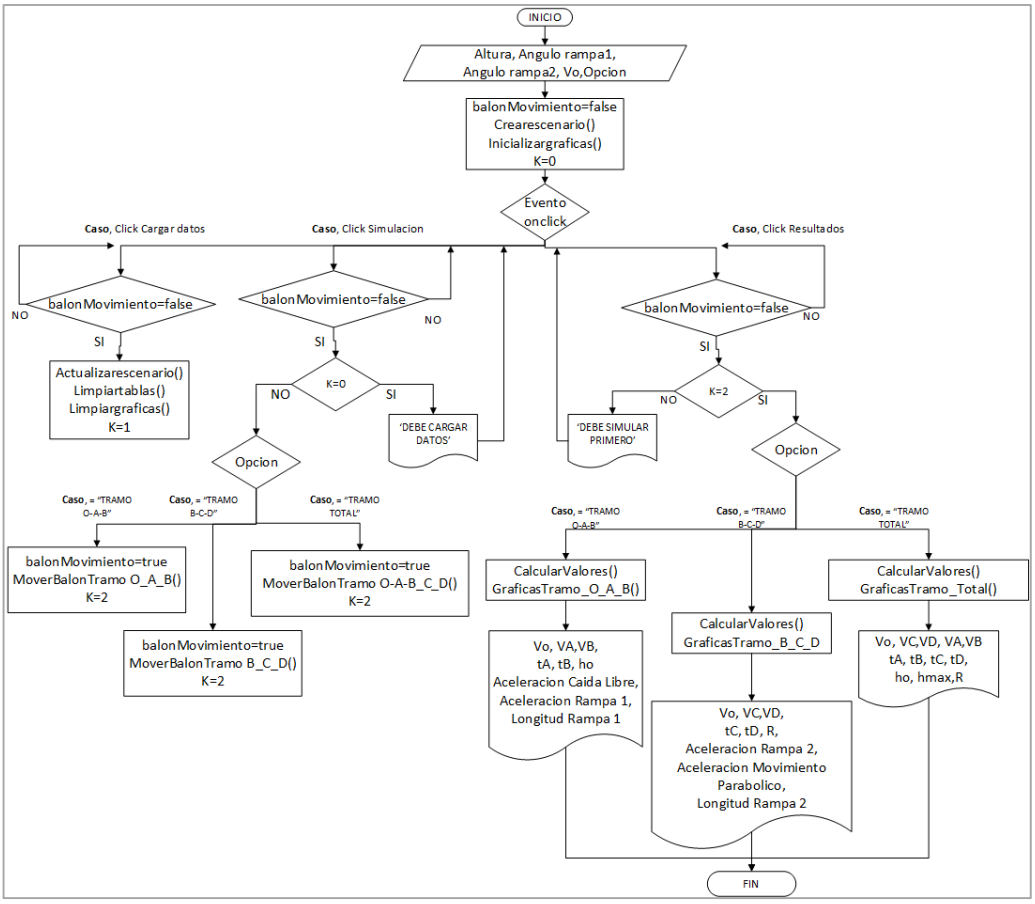

Figura 15. Algoritmo desarrollado para la práctica de movimiento bidimensional.

Similarmente al algoritmo definido para la práctica de vectores, el comportamiento de la aplicación va a estar controlado por los botones definidos para este caso (cargar datos, simulación y resultados). Por medio de un condicional se evalúa el evento *onclick*, dando paso a diferentes procesos y acciones, los cuales se describen a continuación:

A. Caso 2: Click en el botón "Cargar datos"

Para ingresar a las funciones asociadas a este boton y a los otros dos es necesario que el balón no se encuentre en movimiento y que por tanto no este en proceso ninguna animacion. Esto se controla mediante la variable $balonMovimiento$, cuyo valor debe estar en *false*, de este modo las funciones asociadas a estos botones podrán ser llamadas. Las funciones en cuestión son: **Actualizarescenario()**, **limpiartablas()** y **LimpiarGraficas()**, estas permiten la variación del ecenario con

base en los datos de entrada que se ingresen y que además se limpien las tablas y las gráficas de simulaciones pasadas. La variable k toma el valor de 1 para indicar al sistema que los datos fueron cargados.

B. Caso 1: Click en el botón "Simulación"

De forma similar, si la variable $balonMovimiento$ toma el valor de true este botón no tiene ningún efecto, pero si toma el valor de false se sigue el procedimiento indicado en la figura 15. Para que se lleva a cabo la simulación es necesario cargar los datos de entrada para actualizar el escenario, lo cual es verificado a través de la variable k, ya que esta toma un valor de "0" cada vez que se realiza un cambio en la entrada de datos y toma el valor de "1" cuando estos han sido cargados. De esta manera si $k = 0$ se debe desplegar un aviso que indique la necesidad de cargar los datos para poder realizar la simulación, de lo contrario el sistema hará que dicho proceso no se pueda realizar.

En caso contrario ($k = 1$) se tienen tres opciones de simulación definidos a través de la variable "Opción", los cuales son: Opción 1 (tramo O-A-B), la pelota realiza el movimiento de caída libre y de traslación por la rampa 1. Opción 2 (tramo B-C-D), el balón parte del inicio de la rampa 2 y realiza un movimiento parabólico hasta caer en un punto dado al nivel del suelo. Opción 3 (tramo total), la pelota recorre todo el trayecto.

De esta forma, dependiendo de la opción escogida el balón realiza su desplazamiento por los respectivos trayectos del sistema y la variable $balonMovimiento$ toma el valor *true*. Cuando finalice el recorrido de la pelota dicha variable vuelve a tomar el valor de *false*. En este proceso la variable k también puede tomar el valor de "2" para indicar que ya se realizó la simulación, lo que permite el despliegue de resultados.

C. Caso 3: Click en el botón "Resultados"

Este botón solo permite la visualización de resultados solo si la variable $balonMovimiento = false$ y la variable $k = 2$. Si k es diferente de "2" el sistema debe desplegar un aviso informando que se requiere realizar la simulación primero, en caso contrario, dependiendo de la opción de simulación dada se llaman a las funciones encargadas de calcular los valores numéricos y de realizar las gráficas.

2.5.2.1. Ecuaciones de movimiento

Con base en el movimiento de la pelota a través del tramo total, el sistema despliega una serie de resultados numéricos y gráficos asociados con cada uno de los segmentos que componen el sistema completo. Por ejemplo, las expresiones matemáticas utilizadas para describir las curvas de movimiento correspondientes a la posición en x (x vs t), posición en y (y vs t), aceleración (a vs t) y velocidad (v vs t) respecto al tiempo vienen dadas de la siguiente manera:

Ecuaciones de aceleración a para los cuatro tramos del sistema (tramo O-A, tramo A-B, tramo B-C y tramo C-D respectivamente), considerando que en el tramo de caída libre y de movimiento parabólico dicha aceleración es la misma y está representada por medio de la ecuación (11):

$$a_y = -g \tag{11}$$

$$a_x = g\sin\theta_1 \tag{12}$$

$$a_x = -g\sin\theta_2 \tag{13}$$

Ecuaciones de velocidad v para los cuatro tramos respectivamente:

$$v = -gt \tag{14}$$

$$v = v_A + g\sin\theta_1 t \tag{15}$$

$$v = v_B - g\sin\theta_2 t \tag{16}$$

$$v = \sqrt{\left(v_C\cos\theta_2\right)^2 + \left(-gt + v_C\sin\theta_2\right)^2} \tag{17}$$

Ecuaciones de posición en x para los cuatro tramos respectivamente:

$$x = 0 \tag{18}$$

$$x = \left(\frac{1}{2}g\sin\theta_1 t^2 + v_A t\right)\cos\theta_1 \tag{19}$$

$$x = \left(-\frac{1}{2} g \sin\theta_2 t^2 + v_B t \right) \cos\theta_2 \quad \textbf{(20)}$$

$$x = v_C \cos\theta_2 t \quad \textbf{(21)}$$

Ecuaciones de posición en y para los cuatro tramos respectivamente:

$$y = \frac{-gt^2}{2} + h \quad \textbf{(22)}$$

$$y = \left(\frac{1}{2} g \sin\theta_1 t^2 + v_A t \right) \sin\theta_1 \quad \textbf{(23)}$$

$$y = \left(-\frac{1}{2} g \sin\theta_2 t^2 + v_B t \right) \sin\theta_2 \quad \textbf{(24)}$$

$$y = -\frac{1}{2} g t^2 + v_C \sin\theta_2 t + h_{r2} \quad \textbf{(25)}$$

Donde g corresponde al valor de la gravedad, θ_1 y θ_2 son los ángulos de elevación de las rampas 1 y 2 respectivamente, v_A, v_B y v_C son las velocidades de la pelota en los puntos A, B y C, h es la altura inicial de la pelota y h_{r2} es la altura de la rampa 2.

Nota: las gráficas correspondientes a las ecuaciones (11) hasta la (25) se muestran en el capítulo 3 del presente trabajo. Por su parte, el desarrollo completo de las expresiones físicas de la práctica de movimiento bidimensional se encuentra disponible en el numeral 2 del anexo A (ecuaciones de movimiento de los sistemas físicos diseñados).

2.5.3. Proceso de codificación: Práctica de movimiento en dos dimensiones

La simulación del sistema diseñado para la práctica de movimiento bidimensional se basa también en la utilización del framework Phaser, por lo tanto su desarrollo sigue una estructura similar al realizado en la práctica de vectores, con la única variante de que para este caso serán implementadas algunas gráficas de

movimiento que sirvan como complemento a los resultados numéricos obtenidos en cada una de las actividades de simulación realizadas.

Como punto inicial se cambia la relación metros – pixeles para la conformación del escenario de simulación de acuerdo a las características del sistema en el que está basado la presente práctica. Por lo tanto, realizando el respectivo proceso de conversión de unidades se tiene que en este caso 100 metros equivalen a 44.83 pixeles. Por otra parte, como la práctica se divide en tres actividades de simulación de acuerdo a los tramos en los que fue seccionado el sistema (tramo O-A-B, tramo B-C-D y tramo total), se debe implementar dentro del archivo HTML la opción de seleccionar alguna de estas tres alternativas de simulación. Para ello se utiliza la etiqueta <select>, como se muestra en la figura 16:

```
<select id='selec'>
    <option value="tramo-OB">Tramo O-A-B</option>
    <option value="tramo-BD">Tramo B-C-D</option>
    <option value="total">Tramo Total</option>
 </select>
```

Figura 16. Creación de la etiqueta select

Para hacer uso de las tres opciones de simulación, dentro del archivo js se accede al identificador del elemento select mediante document.getElementById (“selec”) y a través de la segunda línea de código de la figura 17 la variable opción toma el valor que se haya seleccionado.

```
eleccion=document.getElementById("selec")
opcion= eleccion.options[eleccion.selectedIndex].text;
```

Figura 17. Creación de variable que obtiene el valor seleccionado en la etiqueta select.

Ya que cada simulación generalmente utiliza datos de entrada diferentes, se debe contar con una función que actualice el escenario cada vez que los parámetros iniciales sean cambiados. Por medio de la función **Create ()** se llama a otra función denominada **cargardatos ()** que es la que lleva a cabo este proceso. La estructura de la función Create se muestra en la figura 18:

```
create: function () {

this.cargardatos()

document.getElementById('Enviar').onclick = function (ev) {
        if (!this.enMovimiento) {
        this.bmd.clear()

        this.cargardatos()
    }
  }.bind(this)

  document.getElementById('Enviar1').onclick = function (ev) {
        if (!this.enMovimiento) {
        this.mover_balon_caida()

        }
  }.bind(this)

  document.getElementById('Enviar2').onclick = function (ev) {
        if (!this.enMovimiento) {
        this.resultados()

        }
  }.bind(this)

  this.inicializar_graficas()

},
```

Figura 18. Definición de la función Create ().

Por medio de esta función se definen los botones cargar datos, simulación y resultados, los cuales se van a utilizar para realizar el proceso completo en cada simulación. Junto a ellos se han definido los identificadores enviar, enviar1 y enviar2 respectivamente, de modo que cuando alguno de los botones es presionado por el usuario, el sistema internamente llamará a la función que esté asociada con cada uno de ellos.

La función **cargardatos ()** es la encargada de manejar lo referente a la creación del fondo para el escenario de simulación, la colocación de imágenes en su interior y la implementación de las secuencias de animación a través de los respectivos sprites. Para el caso del fondo utilizado, su proceso de creación es similar al realizado dentro de práctica de vectores, con la diferencia de que en esta ocasión se creó un gradiente compuesto por cuatro colores. De acuerdo al diseño del sistema se requiere la implementación de dos planos o rampas inclinadas por los cuales pueda desplazarse el cuerpo que va a estar en movimiento, para ello, se recurre al objeto BitmapData, el cual permite el acceso a las funcionalidades del Api de Canvas.

Los métodos utilizados para este caso fueron los siguientes: **beginPath ()**, **moveTo (x, y)**, **lineTo (x, y)**, **fill ()**, **stroke ()** y la manera en que fueron empleados para trazar la forma básica de la rampa se muestra en la figura 19:

```
ctx.beginPath();
var grd1 = ctx.createLinearGradient(dist_inicio_rampa1_x,game.height-alt
grd1.addColorStop(0.1, '#9F9265');
grd1.addColorStop(0.9, '#8F8354');
ctx.fillStyle= grd1;
ctx.strokeStyle="white";
ctx.moveTo(dist_inicio_rampa1_x ,dist_inicio_rampa1y2_y);
ctx.lineTo(dist_inicio_rampa1_x,dist_inicio_rampa1y2_y-altura_rampa1);
ctx.lineTo(base_rampas+dist_inicio_rampa1_x,dist_inicio_rampa1y2_y);
ctx.fill();
ctx.stroke();
```

Figura 19. Creación de un tramo de la rampa1.

La secuencia de pasos incluida dentro de la estructura del código mostrada en la gráfica se resume de la siguiente manera:

a) Por medio del método **beginPath ()** se indica que se va a iniciar una nueva figura.

b) Se crea el gradiente con el cual se va a rellenar el interior de la figura que está por crearse.

c) Se definen las propiedades fillStyle y strokeStyle para declarar los colores del interior de la figura y de su contorno respectivamente.

d) Se inicia el trazado de la figura correspondiente a un triángulo, para ello se coloca el lápiz de dibujo sobre un punto específico mediante el método **moveTo (x, y)** como se indica en la línea de código número siete.

e) Posteriormente, con el método **lineTo (x, y)** se traza una línea desde la posición actual del lápiz (indicada con moveTo) hasta la posición que indican los atributos x y y del método **lineTo ()**, pasando a ser esta la posición actual del lápiz. Si se desea trazar otra línea se emplea nuevamente el método **lineTo ()**.

f) Finalmente se rellena la figura con el método **fill()** y se traza el contorno con el método **stroke()**, con lo cual queda definido gráficamente el triángulo inferior, como se observa en la figura 20:

Figura 20. Diseño de la rampa 1.

g) Posteriormente se emplearon los mismos métodos para dibujar una sección complementaria de la rampa que le diera un aspecto de profundidad y tridimensionalidad.

Aparte de las rampas, la simulación involucra dos sprites para generar la secuencia de animación. El primero de ellos está compuesto por seis imágenes del cuerpo que realiza los movimientos en los cuatro tramos a lo largo del escenario, que en este caso corresponde a un balón de basquetbol (ver figura 21).

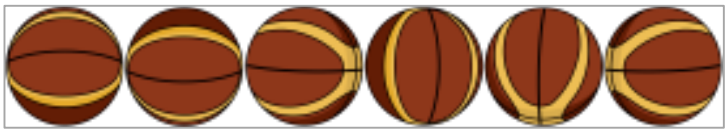

Figura 21. Sprite del balón empleada en la practica 2.

El segundo sprite está conformado por dos imágenes de un globo aerostático cuya función dentro del sistema es la de iniciar el movimiento de caída libre de la pelota desde el punto de altura inicial ingresado por el usuario (ver figura 22).

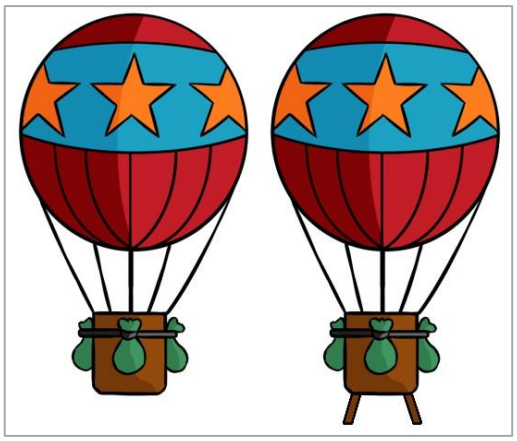

Figura 22. Sprite del globo empleada en la practica 2.

En cuanto a los movimientos realizados por la pelota, se tienen tres casos de simulación para calcular los puntos de las trayectorias en cada uno de ellos. En general, el método utilizado es el de interpolación lineal, sin embargo para la trayectoria parabólica se emplean las ecuaciones de posición en x y y del movimiento parabólico de la pelota (tramo C-D del sistema general). La figura 23 muestra la estructura de código que corresponde a la trayectoria parabólica:

```
for (var q = 0; q <= td; q+=y ) {

    var ox=  (vox*q)*diezU/10+(base_rampas*2+dist_inicio_rampa1_x)
    var oy=  dist_inicio_rampa1y2_y-(((-9.8*Math.pow(q,2))/2)+voy*q+hrampa2)*diezU/10-balon.height
    this.camino1.push({ x: ox, y: oy })

}
```

Figura 23. Código para encontrar los puntos de la trayectoria parabólica.

En ella se observa que por medio de un ciclo for se evalúan las funciones desde el instante cero hasta el tiempo t_D que tarda la pelota en completar la trayectoria parabólica y los valores de las funciones obtenidos en cada punto del recorrido se van guardando en un arreglo llamado this.camino1 a través del método **push ()**. Para acceder a estos puntos y asignarlos posteriormente a la posición del balón se debe hacerlo a través de la función **Update ()** como se indicó en la simulación de vectores.

En cuanto a las gráficas de movimiento de la pelota se recurre a la librería Epoch, la cual es agregada inicialmente dentro del archivo HTML como se muestra en la figura 24:

```
<script type="text/javascript" src="librerias/d3/d3.min.js"></script>
<script type="text/javascript"  src="librerias/epoc/epoch.min.js"></script>
```

Figura 24. Librería epoch agregada al archivo HTML.

Seguidamente se debe crear un que contenga la respectiva gráfica, en el cual, aparte de ser definidas las dimensiones de la misma, se especifica la clase, que para este caso corresponde a epoch. Este procedimiento se muestra en la figura 25:

```
<h3 id="x">x(m)</h3>
<div id="grafica1">
    <div id='primero'>x vs t</div>
    <div id="grafica_x_vs_t" style="width: 369.92; height: 359.64" class="epoch" ></div>
    <h3 id="t">t(s)</h3>
</div>
```

Figura 25. Código para agregar la gráfica en el archivo HTML.

En el archivo js se crea un arreglo de objetos dentro del cual se guardan los datos que se van a mostrar a través de las gráficas que se vayan a implementar. Por ejemplo, en la figura 26 se muestra el proceso de creación de la variable que va a contener los datos para la primera de las gráficas de movimiento: posición en x contra tiempo (x vs t). En ella se observa que fueron agregados cuatro objetos, correspondientes con los cuatro tramos en los que ha sido dividido el sistema (tramos O–A, A–B, B–C y C-D). Posteriormente por medio de la propiedad values se cargan los datos para los dos ejes de la gráfica.

```
var datos_x_vs_t = [
   {values: []},
   {values: []},
   {values: []},
   {values: []}
   ];
```

Figura 26. Arreglo para cargar los valores de las gráficas.

El paso siguiente es implementar una función que permita iniciar el sistema de ejes coordenados sobre los cuales van a ir dibujadas las gráficas cada vez que la página donde se aloja la simulación sea cargada. Por ello se define la función denominada **Inicializar Graficas ()**, la cual puede ser llamada a través de la función predefinida Create, y cuya estructura se muestra en la figura 27. En ella se especifica el identificador del div al cual se desea enviar la gráfica, el tipo de gráfica, el nombre de la variable que contiene los datos, los ejes empleados y las márgenes que delimitan el área para cada gráfica. El código solo muestra el fragmento en el que se definió la gráfica de x vs t, sin embargo las otras gráficas se inicializan dentro de la misma función:

```
inicializar_graficas: function(){

    grafica_x_vs_t = $('#grafica_x_vs_t').epoch({ type: 'line', data: datos_x_vs_t, axes: ['left', 'bottom'],
                    margins: { top: 10, right: 10, bottom: 20, left: 30 }});
},
```

Figura 27. Ejemplo de cómo iniciar las gráficas.

A partir de este punto ya se pueden crear las funciones para definir cada uno de los valores que se desean graficar. Por ejemplo la figura 28 muestra la manera en que son asignados los valores al primer elemento de las variables encargadas de guardar los datos, que para este caso corresponden a aquellos que permiten graficar las curvas de posición en x contra tiempo (x vs t), posición en y contra tiempo (y vs t), velocidad contra tiempo (v vs t) y aceleración contra tiempo (a vs t) para el primer tramo del recorrido total de la pelota (tramo O-A), es decir el correspondiente al movimiento de caída libre.

```
for (var taa = 0; taa <= ta; taa += 0.01) {

    datos_x_vs_t[0].values.push({x: taa, y: 0});
    datos_y_vs_t[0].values.push({x: taa, y: (((-9.8*Math.pow(taa,2))/2)+htotal)});
    datos_v_vs_t[0].values.push({x: taa, y: 9.8*taa});
    datos_a_vs_t[0].values.push({x: taa, y: 9.8});

    }
```

Figura 28. Ejemplo de cómo cargar los datos que permiten graficar el tramo O-A.

Finalmente, al haber definido los datos necesarios para realizar las curvas de movimiento, ya se pueden cargar para ser graficados mediante el método **Update ()**, pero antes se debe definir el rango y el dominio de los valores permitidos dentro de las gráficas con el método **option ()**. La figura 29 muestra la configuración realizada para la primera gráfica (x vs t):

```
grafica_x_vs_t.option('ticks', { top: 0, right: 0, bottom: 8, left: 8 })
grafica_x_vs_t.option('range', [0, 40])
grafica_x_vs_t.option('domain', [0, Math.ceil(ta+tb)])
grafica_x_vs_t.update(datos_x_vs_t)
```

Figura 29. Forma de cargar los datos a gráficas.

2.6. PRÁCTICA 3 - MOVIMIENTO CIRCULAR

2.6.1. Diseño del sistema

El esquema general del sistema propuesto para la práctica de movimiento circular se compone de una pista de acrobacias formada por dos segmentos, uno de tipo rectilíneo y otro circular, a través de los cuales un motociclista experto en maniobras debe transitar con el objetivo de realizar un giro completo, como se muestra en la figura 30. El tramo rectilíneo cuenta con una distancia variable d_1, medida desde el punto de partida de la pista (punto O) hasta la proyección inferior del centro del circulo (punto A). Por su parte el segmento circular, de radio variable r, se ubica en un punto específico de la pista contiguo al tramo rectilíneo.

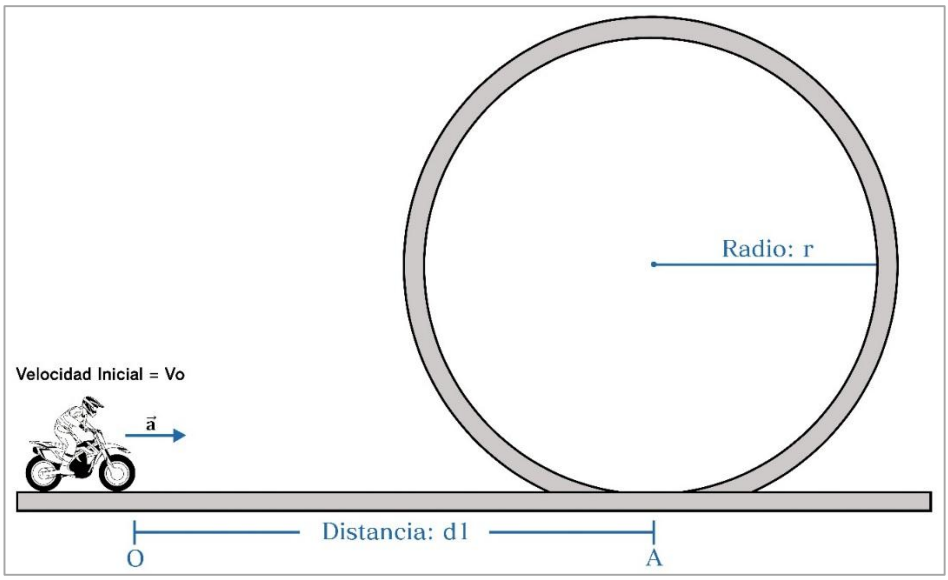

Figura 30. Práctica de movimiento circular – Sistema general.

La dinámica del sistema permite que el motociclista inicie su carrera con una velocidad inicial cualquiera (v_0) y se desplace por el segmento rectilíneo bajo la acción de una aceleración constante ($\vec{a}$) hasta alcanzar el final de dicho tramo. Dependiendo de la velocidad con que el acróbata llegue al punto A podrá dar un giro completo dentro de la circunferencia sin caer en el intento. El movimiento del piloto dentro del segmento circular estará definido por las fuerzas que actúan sobre él en cada punto del tramo. Es importante aclarar que las superficies de ambos segmentos poseen igual coeficiente de fricción.

2.6.1.1. Objetivo de la práctica de movimiento circular

Validar los conceptos de aceleración radial y tangencial de un cuerpo que se desplaza dentro de un sistema dinámico con movimiento circular, considerando los efectos causados por las fuerzas que actúan sobre él.

La práctica permite al estudiante comprobar matemáticamente las funciones de velocidad, posición y aceleración del piloto al final del segmento lineal, de modo que se pueda establecer una relación de velocidad al inicio del segmento circular para determinar si este realiza un giro completo. En cuanto al movimiento a través de la circunferencia, se podrán corroborar las funciones generales de aceleración tangencial (a_t) y radial (a_r), así como también las de velocidad y posición en términos del ángulo recorrido.

2.6.2. Diseño de la aplicación

El algoritmo diseñado para la implementación de la práctica de movimiento circular se muestra en la figura 31. En él se observa que los datos de entrada corresponden a la velocidad inicial del piloto en el tramo recto (v_0), la distancia de separación del punto de partida con la entrada al tramo circular ($Distancia$), el radio de esta circunferencia ($Radio$) y el coeficiente de fricción de toda la superficie de la pista (miu).

De modo similar a los otros algoritmos se tiene un proceso inicial donde se llama a la función encargada de crear el escenario compuesto por el fondo, las imágenes y el sprite de un motociclista que es el encargado de realizar los movimientos en los trayectos definidos del sistema. También es llamada la función **inicializargraficas ()**, con la cual se adicionan los ejes coordenados para la representación de las gráficas. Posteriormente se analiza la variable $enMovimiento$ con un valor de *false* que será la encargada de indicar cuando el piloto esté en movimiento. Finalmente se define la variable k con una valor inicial de “0” para informar al sistema que ha habido ingreso de datos de entrada. Esta variable cumple las mismas funciones que en el movimiento en 2 dimensiones.

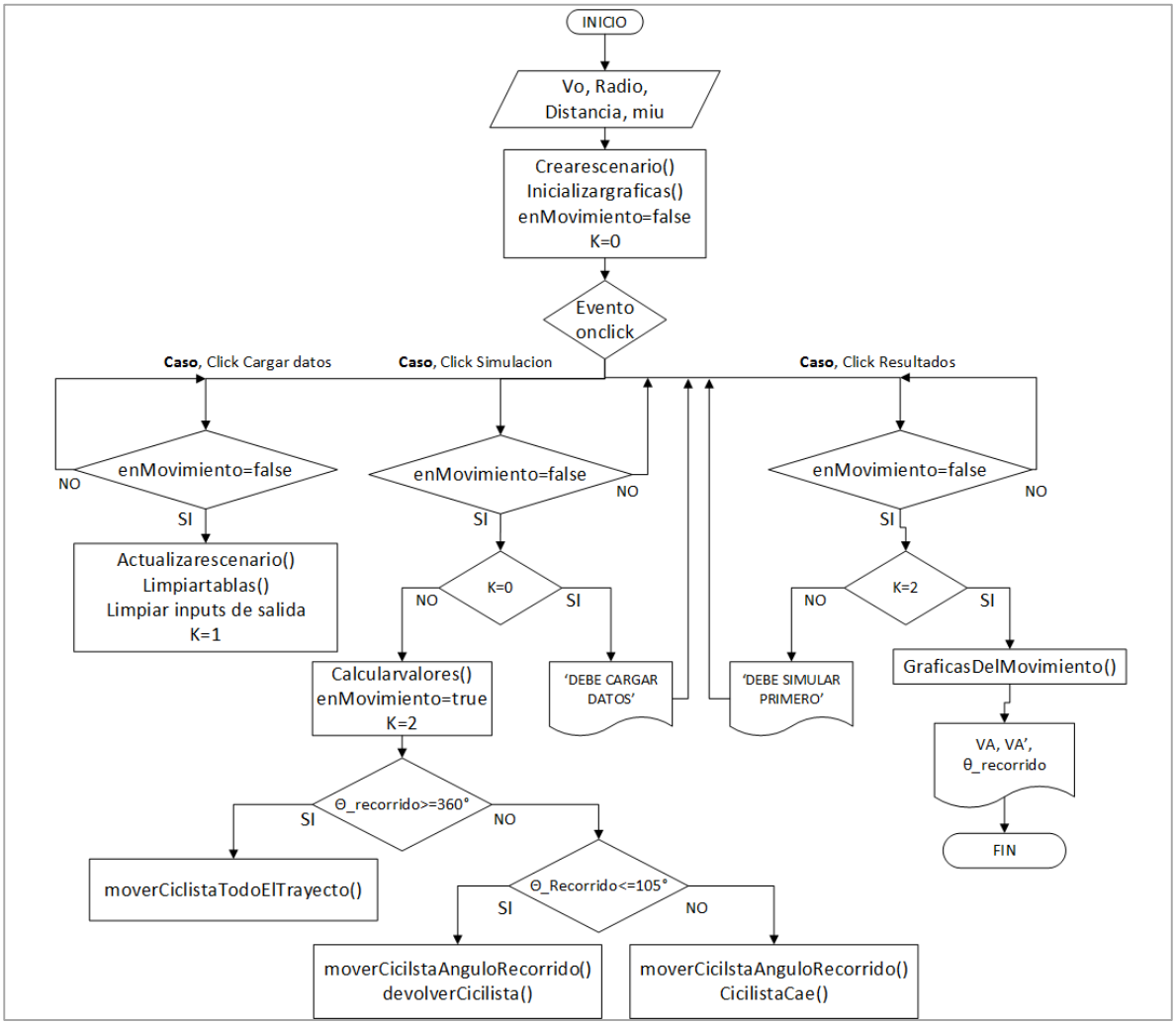

Figura 31. Algoritmo de la práctica de movimiento circular.

En cuanto a los botones, la simulación cuenta con los mismos tres que en el caso anterior (cargar datos, simulación y resultados) y cumplen idéntica función de controlar el comportamiento de la aplicación. Su acción es evaluada mediante un condicional que define en que momento alguno de ellos es presionado (evento onclick) como se muestra a continuación:

A. Caso 1: Click en botón "Cargar datos"

Si el piloto está en movimiento este botón no tiene efecto, en caso contrario, de haber variación en los datos de entrada el escenario será actualizado, se limpian las gráficas de simulaciones anteriores y se limpian los *inputs* donde se muestran los datos de salida (resultados numéricos). También la variable k toma el valor "1"

para indicar que los datos han sido cargados de modo que se pueda llevar a cabo el proceso de simulación.

B. Caso 2: Click en botón “Simulación”

Si hay una variación de datos y estos no han sido cargados, la variable k toma el valor de “0” y no se permite la simulación. En este caso debe haber un aviso que informe que hay que realizar el proceso de carga de esos datos para poder actualizar el escenario. En caso contrario se tiene un proceso donde es llamada la función de calcular valores con el fin de conocer el ángulo que recorre el motociclista, además la variable $enMovimiento$ toma el valor *true*, dando a entender que se deben activar las respectivas secuencias de animación. La variable k ahora toma el valor de “2” de modo que es posible desplegar los resultados de la simulación.

Dependiendo del ángulo recorrido por el motociclista se tienen tres casos:

- Si el ángulo recorrido es igual a 360°, la simulación muestra la secuencia en que el piloto realiza un giro completo por el tramo circular, obteniendo una velocidad final diferente de cero.

- Si el ángulo recorrido es mayor que 105° y menor que 360° la velocidad final del motociclista es cero y por tanto se muestra la secuencia correspondiente a la caída de dicho piloto desde la plataforma circular.

- Si el ángulo recorrido es menor que 105° el piloto llega a una velocidad final igual a cero y se devuelve por la plataforma circular hasta llegar al inicio de esta.

C. Caso 3: Click en botón “resultados”

Cuando se hace click en este botón se tiene un proceso designado para mostrar las gráficas de movimiento producto de la simulación: velocidad, aceleraciones y la fuerza normal (N) con respecto al ángulo θ en el segmento circular. Por su parte en el tramo lineal las gráficas realizadas son las de la velocidad y aceleración con respecto al tiempo. Como resultados numéricos se tienen: el ángulo calculado ($\theta_{Recorrido}$) y las velocidades del piloto cuando al inicio y al final del movimiento circular (v_A y $v_A{}'$ respectivamente).

2.6.2.1. Ecuaciones de movimiento

De acuerdo a la figura 30 se definen las expresiones generales que representan el movimiento del piloto a través de los dos tramos que componen el sistema. Por ejemplo, las ecuaciones de velocidad (v), de aceleración tangencial (a_t), de aceleración radial (a_r) y de aceleración total (a) del motociclista dentro de la circunferencia se muestran a continuación:

$$v = \sqrt{\frac{v_0^{\,2} - 2\theta_{rad} rg \sin\left(\theta - \frac{\pi}{2}\right) - 2\theta_{rad}\mu rg \cos\left(\theta - \frac{\pi}{2}\right)}{1 + 2\theta_{rad}\mu}} \quad \textbf{(26)}$$

$$a_t = -g \sin\left(\theta - \frac{\pi}{2}\right) - \frac{\mu N}{m} \quad \textbf{(27)}$$

$$a_r = \frac{N}{m} - g \cos\left(\theta - \frac{\pi}{2}\right) \quad \textbf{(28)}$$

$$a = \sqrt{a_t^{\,2} + a_r^{\,2}} \quad \textbf{(29)}$$

Donde θ corresponde al ángulo recorrido dentro de la plataforma circular, μ es el coeficiente de fricción de la superficie de contacto, N es la fuerza normal y r al radio de la circunferencia.

Nota: Las gráficas de movimiento correspondientes a las ecuaciones (26), (27), (28) y (29) se muestran en el capítulo 3. Por su parte, el desarrollo completo de las expresiones físicas de la práctica de movimiento circular se encuentra en el numeral 3 del anexo A (ecuaciones de movimiento de los sistemas físicos diseñados).

2.6.3. Proceso de codificación: Práctica de movimiento circular

La práctica de movimiento circular sigue la estructura de desarrollo de Phaser. Por lo tanto en las funciones **Preload ()**, **Create ()** y **Update ()** llevan a cabo procedimientos similares a los desarrollados en las prácticas de vectores y de movimiento en dos dimensiones.

En el caso de la implementación del fondo del área de simulación se creó un gradiente con cuatro colores sobre el cual fueron adicionadas algunas imágenes complementarias a la estética de la pista de acrobacias por donde transita el motociclista, las cuales fueron implementadas por medio de la herramienta de dibujo digital Medibang Paint Pro. En cuanto a la plataforma circular, esta fue realizada a través del método de Canvas **arc (x, y, radio, ángulo inicio, ángulo final, dirección),** el cual genera un arco cuyo centro está dado por la posición (x, y), con un radio específico y creado desde un ángulo determinado por los atributos del método. En la figura 32 se observa el código que detalla la creación del segmento circular:

```
ctx.beginPath();
ctx.lineWidth=18
var grd1 = ctx.createLinearGradient(longitud,game.height-diezU/2.2-radio*2,longitud,game.height-diezU/3);
grd1.addColorStop(0.5, '#7F6F51');
grd1.addColorStop(0.6, '#99855F');
ctx.fillStyle=grd1;
ctx.arc(longitud+moto.width+distanciaInicioMoto_x,distanciaInicioMoto_y-radio,radio+anchorCirulo,0,Math.PI*2,true);
ctx.strokeStyle=grd1
ctx.stroke();
```

Figura 32. Código con el cual se crea la plataforma circular.

En términos generales se siguen los mismos pasos para la creación de figuras con el Api de Canvas, sin embargo la variante es el método **arc ()**, donde se aprecia que la componente x del centro del círculo que se desea crear va a depender del radio del mismo. Los otros parámetros importantes para la conformación del segmento circular corresponden a los ángulos de inicio y de final del arco (círculo), en donde se especifican los valores de 0 para el inicio del tramo circular y 2π para el final del mismo. El último parámetro del método arc () corresponde a un valor booleano con el cual se especifica si la dirección de conformación del arco se realiza a favor o en contra de las manecillas del reloj.

Por otra parte, para establecer el tamaño del trazo con el que se dibuja el círculo se utiliza la propiedad lineWidth, cuyo parámetro indica precisamente que tan ancho o delgado es el contorno del tramo circular. El valor estándar predefinido es lineWidth = 1, cuyo trazo es muy fino, sin embargo para el círculo implementado se tomó un valor de lineWidth = 18 ya que se requería un tipo de plataforma circular con ciertas dimensiones, acordes con el diseño real de este tipo de pistas de acrobacias. También, dentro del círculo se crearon 10 arcos distribuidos de manera simétrica, con diferentes ángulos de inicio y final para cada uno, con el fin de darle una mejor apariencia a la plataforma circular creada.

En cuanto al sprite utilizado para generar la secuencia de animación del movimiento del motociclista está conformado por 5 cuadros, de los cuales los tres primeros se utilizan para dar movimiento a las ruedas de la moto y los dos restantes son usados para el caso en que el motociclista no realiza el giro completo y cae (ver figura 33).

Figura 33. Sprite con la secuencia de movimientos del piloto.

El recorrido del motociclista a través de los puntos de la trayectoria que se vayan a definir se realiza mediante el método de interpolación lineal. Para determinar estos puntos en el tramo lineal se lleva a cabo un procedimiento similar al de la práctica de vectores, sin embargo, dentro del tramo circular se aplica una lógica distinta, ya que los puntos que representan su traslación dependen del ángulo que en teoría alcanza a recorrer el motociclista con base en los datos iniciales dados por el usuario. La condición para determinar el ángulo que recorre el motociclista dentro de la plataforma circular viene dada por la siguiente expresión:

$$\frac{v_0^{\,2}}{2gr} > \left(sen\left(\theta - \pi/2\right) + \mu\cos\left(\theta - \pi/2\right)\right)\theta \qquad \textbf{(30)}$$

El desarrollo de esta expresión se encuentra está disponible en el numeral 3.1.2. del anexo de ecuaciones de movimiento. Los términos que aparecen dentro de la inecuación (30) son los siguientes:

v_0: Velocidad inicial con la que el motociclista llega al inicio del tramo circular.

g: Gravedad.

r: Radio del tramo circular.

μ: Coeficiente de fricción del trayecto circular.

θ: Angulo recorrido por el piloto medido desde el punto inicial del segmento circular.

La expresión (17) indica que mientras el término de la izquierda de la inecuación sea mayor que la de la derecha la velocidad del piloto dentro del círculo va a ser mayor que cero. Con base en esto se debe encontrar el ángulo barrido por el piloto

para el cual la velocidad se hace cero, de modo que se pueda determinar el punto de la plataforma circular en que este cae. Dicho procedimiento es mostrado en la figura 34:

```
teta1=0
e=(Math.sin(teta1-Math.PI/2)+miu*Math.cos(teta1-Math.PI/2))*teta1
f=Math.pow(velocidad,2)/(2*9.8*r)

while(e<=f && teta1<2*Math.PI) {
        teta1=teta1+0.1*Math.PI/180
        e=(Math.sin(teta1-Math.PI/2)+miu*Math.cos(teta1-Math.PI/2))*teta1
        }

    teta=(Math.round((teta1-0.1*Math.PI/180)*10000)/10000)
```

Figura 34. Código para determinar el ángulo de recorrido del piloto.

En la estructura del código de la figura 34 se tiene que:

- Se inicializa la variable $teta1 = 0$ y se definen f y e como las variables que van a contener la primera y segunda expresión de la anterior inecuación.

- Seguidamente se utiliza el bucle *while* para condicionar al sistema que mientras se cumpla que la segunda expresión sea menor que la primera, la variable *teta1* se vaya incrementando 0.1 radianes. El valor hallado debe ser pasado a radianes.

- La expresión e se va evaluando cada vez que la variable $teta1$ cambie.

- Cuando se dé el caso en que la condición establecida no se cumpla, la variable $teta$ toma el valor que haya alcanzado $teta1$, de esta manera queda determinado el ángulo que alcanza a recorrer el motociclista en su movimiento por el tramo circular. En caso de que se cumpla que la variable e siempre sea menor que f, significa que el piloto recorre toda la circunferencia y por ende el ángulo recorrido es igual a 360° (2π).

Una vez calculado el ángulo de recorrido se pueden hallar los puntos en pixeles por los cuales deberá desplazarse el piloto representado en el sprite. De acuerdo al ángulo alcanzado se deben considerar tres opciones:

- El motociclista recorre todo el tramo circular.

- El motociclista recorre un ángulo menor o igual que 105° ($7\pi/12$), se detinene y realiza un movimiento de retroceso sobre la plataforma circular.

- El motociclista aunque recorre un ángulo mayor a 105° ($7\pi/12$) no completa el giro y cae de la pista circular.

Con base en los tres casos presentados se tiene la estructura de código que permite determinar los puntos de la trayectoria que debe seguir el piloto para realizar el movimiento de tipo circular (ver figura 35):

```
teta2=-Math.PI/2+teta-moto.width/radio

for (var teta5=-Math.PI/2 ; teta5<=teta2 ; teta5 += Math.PI/180) {
 var px5 = radio*Math.cos(teta5)+longitud+moto.width+distanciaInicioMoto_x

 var py5 = -radio*Math.sin(teta5)+distanciaInicioMoto_y-radio

  camino5.push( { x: px5, y: py5 })
}
```

Figura 35. Código para calcular los puntos que recorrerá el sprite de la moto.

En ella se observa lo siguiente:

- Se emplea un ciclo for para evaluar las variables $px5$ y $py5$, las cuales vienen dadas por las ecuaciones de posición del movimiento circular ($x = rcos\theta$ y $y = rsen\theta$). Estas variables fueron adaptadas al sistema debido a que es necesario definir el punto (0,0) en el centro de la circunferencia del segmento circular, de modo que pudiera ser tomado como referencia para determinar el ángulo recorrido por el motociclista. Las coordenadas del punto dadas en pixeles corresponden a (longitud+moto.width+diezU/15, this.game.height-diezU/1.8-radio).

- Se evalúan las variables de posición desde un valor de $-\pi/2$, ángulo desde el cual se inicia el movimiento dentro del tramo circular.

- Cada valor calculado se va guardando en un arreglo por medio del método push ().

Se debe considerar también que a medida que recorre cada punto calculado, el sprite correspondiente a la imagen de la motocicleta debe girar un determinado valor en grados. Para ello en la función **Update ()**, mediante la propiedad rotation, se establece que por cada grado que la moto recorra dentro de la circunferencia realizará también una rotación de un grado correspondiente al sentido de su movimiento circular (movimiento antihorario), como se muestra en la figura 36:

```
moto.rotation+=-1*Math.PI/180
```

Figura 36. Rotación del sprite.

Finalmente, el proceso de simulación dentro de esta práctica también incluye una serie de gráficas de movimiento que complementan e ilustran algunos de los resultados numéricos hallados. Debido a que las gráficas presentan un comportamiento diferente se utiliza el software MatLab para implementar las funciones que las representan y de esta manera realizar un análisis para determinar los rangos y dominios de cada una de ellas y aplicarlos dentro de la lógica de la simulación.

2.7. PRÁCTICA 4 - LEYES DEL MOVIMIENTO

2.7.1. Diseño del sistema

La práctica de leyes del movimiento consta de un sistema formado por un cuerpo de masa m y un resorte con un valor definido de constante elástica (k), los cuales se hallan ubicados sobre un plano o rampa inclinada con ángulo de elevación variable (θ), como se muestra en la figura 37.

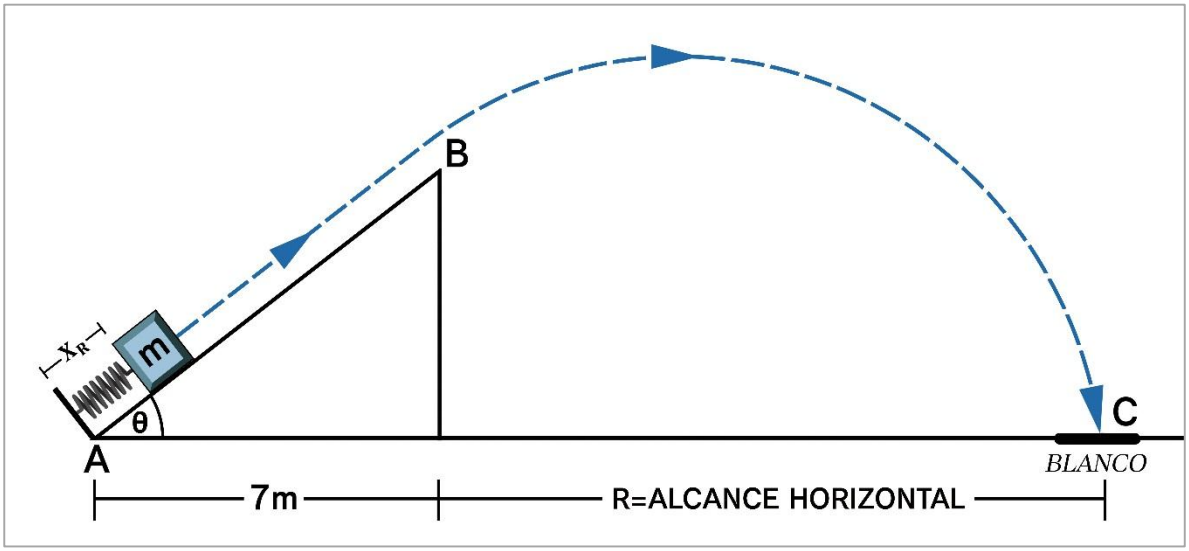

Figura 37. Leyes del movimiento – Sistema general.

El funcionamiento del sistema se ha diseñado de tal forma que al realizar una compresión del resorte, este aplicará sobre el cuerpo una determinada fuerza, provocando en él un movimiento de traslación sobre el plano inclinado. Dependiendo de la longitud de compresión del resorte el cuerpo podrá traspasar los

límites de la rampa y realizar un movimiento de tipo parabólico, consiguiendo diferentes valores de alcance horizontal por fuera del plano inclinado.

El sistema también cuenta con un objetivo o blanco de disparo que puede ser ubicado a diferentes distancias de la base del plano inclinado y dentro del cual se busca introducir el bloque de masa m, partiendo de algunos datos iniciales establecidos como por ejemplo el ángulo de elevación del plano inclinado (θ), la longitud de compresión del resorte ($X_{Resorte}$), el coeficiente de fricción de la rampa (μ) y la distancia de ubicación del blanco (R).

2.7.1.1. Objetivo de la práctica de leyes del movimiento

Validar la relación que existe entre la aceleración producida sobre un cuerpo de masa m y una fuerza neta aplicada sobre él a través de un sistema dinámico dentro del cual se mueve dicho objeto. Por medio de la práctica se puede llevar a cabo la comprobación de los valores numéricos de posición, velocidad y aceleración relacionados con el sistema masa-resorte, la determinación de la longitud de compresión del resorte adecuada para lograr un alcance horizontal deseado, y otra serie de resultados generados también dentro de la práctica de movimiento bidimensional.

2.7.2. Diseño de la aplicación

La figura 38 muestra el algoritmo diseñado para la práctica de leyes de movimiento. Los datos de entrada que se consideran dentro de la simulación corresponden a la longitud de compresión del resorte, el coeficiente de fricción del plano inclinado, el ángulo de inclinación de la rampa y la distancia de ubicación del blanco. En términos de desarrollo, se sigue una lógica similar a la empleada en las prácticas pasadas, con la diferencia de que en este caso la variable que permite determinar los cambios realizados en la entrada de datos se denomina g.

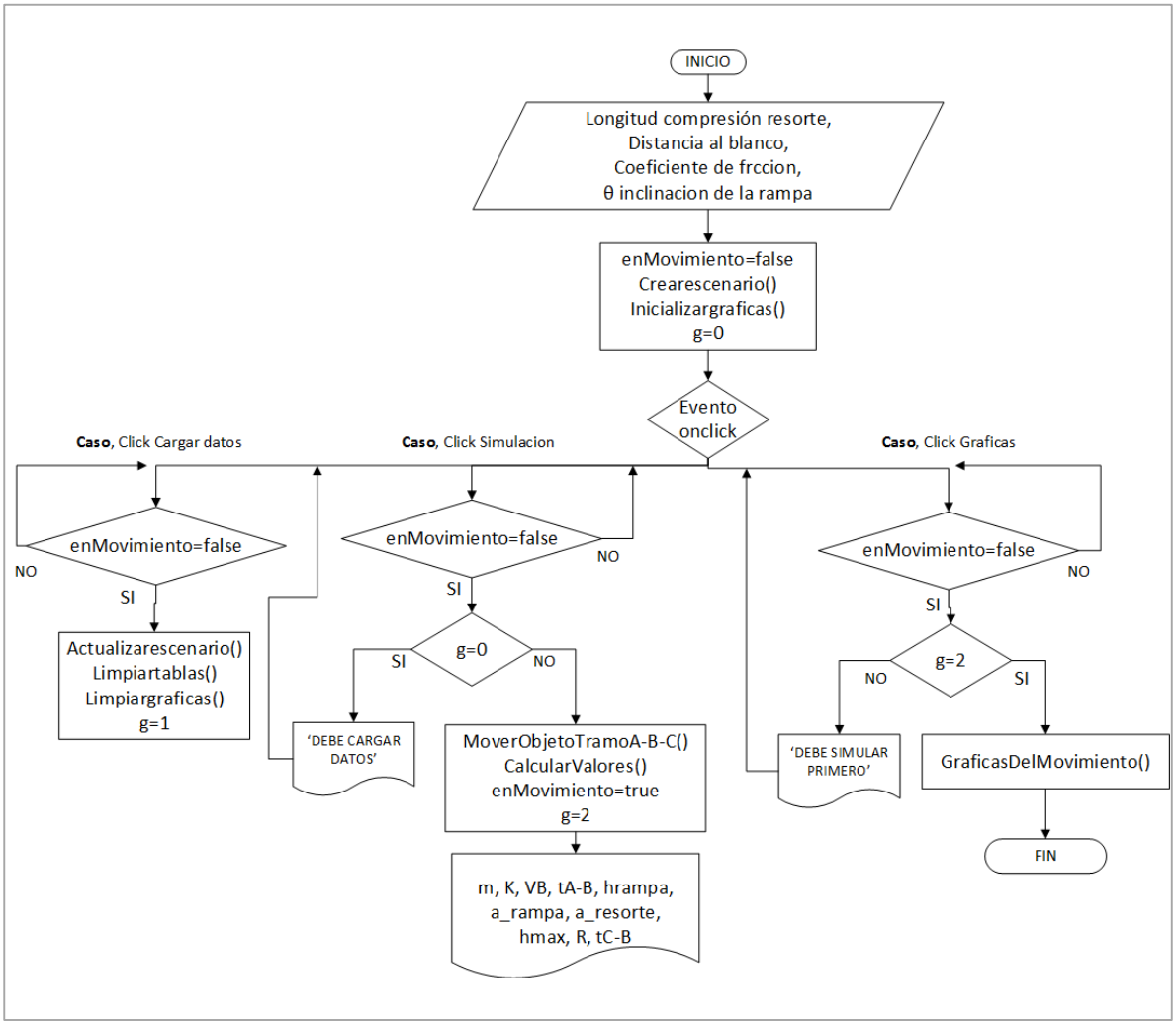

Figura 38. Algoritmo de la práctica de leyes del movimiento.

También se utilizan tres botones para realizar las acciones dentro de la simulación, los cuales se denominan cargar datos, Simulación y Gráficas. Cuando es presionado el botón correspondiente a cargar datos se llevan a cabo los mismos procedimientos que en los algoritmos anteriores. Por su parte cuando se presiona el botón simulación son llamadas automáticamente algunas funciones encargadas de determinar por ejemplo los puntos de la trayectoria que deberá recorrer el bloque, la realización de los cálculos numéricos que permitan mostrar los datos de salida dentro de una tabla.

Dichos resultados son los siguientes: m (masa del bloque), K (constante elástica del resorte), v_B (velocidad del bloque en el punto B), t_{AB} (duración del desplazamiento por la rampa), h_{Rampa} (altura de la rampa), a_{Rampa} (aceleración del bloque dentro de la rampa), $a_{Resorte}$ (aceleración del bloque unido al resorte),

$h_{máx}$ (altura máxima alcanzada en el movimiento parabólico), R (alcance horizontal del bloque en el movimiento parabólico) y t_{BC} (tiempo transcurrido desde el punto B hasta el C).

Después de simular se pueden obtener las respectivas gráficas de movimiento a través del botón “Gráficas”, la simulación desplegará las curvas correspondientes a posición vs tiempo, velocidad del bloque vs tiempo y aceleración vs tiempo.

2.7.2.1. Ecuaciones de movimiento

Con base en el sistema mostrado en la figura 37 se definen las expresiones generales que representan la velocidad del bloque en los tres tramos definidos (tramo A-A’, A’-B y B-C) respectivamente:

$$v = a_{\mathrm{Re}sorte}t = \left[\frac{kx_{\mathrm{Re}sorte}}{m} - g\left(\sin\theta + \mu\cos\theta\right)\right]t \quad \textbf{(31)}$$

$$v = at + v_0 = -g\left(\sin\theta + \mu\cos\theta\right)t + \sqrt{2a_{\mathrm{Re}sorte}x_{\mathrm{Re}sorte}} \quad \textbf{(32)}$$

$$v = \sqrt{\left(-gt + v_B\sin\theta\right)^2 + \left(v_B\cos\theta\right)^2} \quad \textbf{(33)}$$

Donde θ equivale al ángulo de elevación del plano inclinado, $X_{Resorte}$ corresponde a la longitud de compresión del resorte.

Nota: La gráfica correspondiente a la velocidad del bloque dentro de los tres tramos del sistema se muestra en el capítulo 3. El desarrollo completo de las expresiones físicas de la práctica de leyes del movimiento se encuentra en el numeral 4 del anexo A (ecuaciones de movimiento de los sistemas físicos diseñados).

2.7.3. Proceso de codificación: práctica de leyes del movimiento

En esta práctica se sigue el mismo procedimiento para la implementación del fondo y del escenario de simulación a través de los métodos de Canvas, incluyendo la rampa o plano inclinado desde donde se lanza el bloque por medio del resorte. Los demás elementos presentes fueron realizados a partir del software de dibujo digital. Para la secuencia de animación total se emplearon dos tipos diferentes de sprites, uno que representa el movimiento del sistema masa-resorte y otro que corresponde al movimiento del bloque después de que ha abandonado el muelle.

En la figura 39 se muestra el primero de los sprites utilizados, el cual está conformado por 12 cuadros de movimiento que representan la secuencia de movimiento del sistema masa-resorte:

Figura 39. Sprite del sistema masa-resorte.

De manera similar, la figura 40 muestra el segundo de los sprites utilizados el cual está conformado por cinco cuadros de movimiento. La primera imagen del sprite (posición 0) se utiliza para representar el desplazamiento del bloque dentro de la rampa inclinada y en la trayectoria parabólica. Las imágenes de los cuadros restantes corresponden al ingreso del bloque dentro del blanco de disparo, cuando el valor de longitud de compresión del resorte calculada e ingresada en los datos de entrada es la correcta.

Figura 40. Sprite del bloque ingresando en el blanco.

Para coordinar las respectivas secuencias de movimiento generadas con los dos sprites se realiza el siguiente procedimiento: Inicialmente se agrega el sprite de la figura 39 dentro de la función encargada de crear el escenario de simulación, denominada **CargarDatos ()**, como se muestra en la figura 41:

```
resorte = game.add.sprite(distancia_comienzo_rampa_x, punto_en_y_posicioninicial_resorte, 'resorte')
resorte.scale.setTo(game.width / 2200);
resorte.anchor.set(0,1);
resorte.rotation=-teta
resorte.animations.add('uno', [10], 1, true)
resorte.animations.add('dos', [9,8,7,6,5,4,3,2,1,0], 8, true)
resorte.animations.add('tres', [11], 1, true)
resorte.animations.play('uno')
```

Figura 41. Código para agregar el sprite del sistema masa-resorte.

En el código se observan los siguientes procesos:

- En la primera y segunda línea de código se adiciona el sprite y se define el punto de referencia desde donde se inicia el movimiento respectivamente.

- Para que el sprite quede situado sobre la rampa primero debe ser rotado θ grados por medio de la propiedad rotation, explicada en la práctica anterior, y luego se sitúa en el punto indicado. Este valor corresponde al ángulo de

elevación del plano inclinado ingresado por el usuario en el cuadro de datos de entrada.

- Se definen tres animaciones llamadas uno, dos y tres para implementar las respectivas secuencias de movimiento. La animación uno, correspondiente al desplazamiento del bloque unido al resorte, inicia con la imagen de la posición 10 del sprite de la figura 39, la cual muestra el resorte comprimido justo antes de iniciar su movimiento de elongación.

- La animación dos se encarga de recrear el movimiento del sistema masa-resorte desde el punto inicial de compresión hasta cuando el muelle alcanza su posición de reposo y libera al bloque unido a él. Esta secuencia se realiza con las imágenes de los cuadros restantes, comenzando con la posición 9 del sprite y llegando hasta la posición 0, punto en el que el bloque está por abandonar el resorte que lo impulsa.

- Finalmente la animación tres muestra al resorte solo sin el bloque, imagen que corresponde al cuadro final del sprite.

Cuando el botón de simulación es presionado inmediatamente se activa la animación dos en la función **Update ()**. Una vez que la secuencia se haya completado, el sistema da paso a la animación tres, cuya imagen muestra al resorte en su posición de reposo. En este punto se debe incorporar la secuencia del bloque desplazándose a través de la rampa inclinada, para ello se debe determinar su posición dentro de la imagen del sprite que muestra al resorte en posición de equilibrio.

Considerando el triángulo interno formado dentro del plano inclinado con base en la longitud del sprite, como se muestra 42, se puede realizar una aproximación para ubicar la imagen del bloque contigua a la del resorte, de modo que las secuencias de animación muestren continuidad.

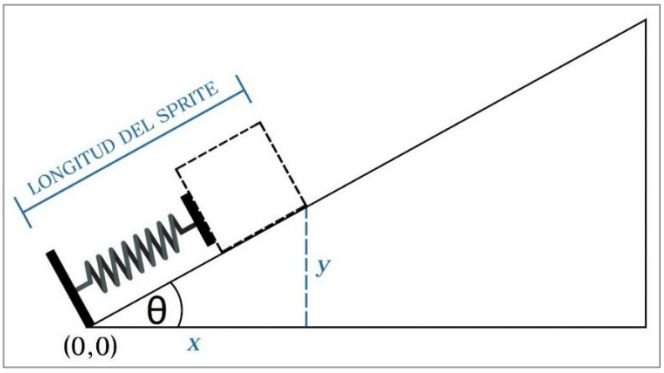

Figura 42. Ubicación del segundo sprite de animación.

Suponiendo que sobre la hipotenusa del triángulo descrito el resorte abarca un 80% de esta y que la posición (0,0) está en el inicio de la rampa, la posición donde se ubica el bloque queda definida como:

*(Math.cos (teta) * resorte.width * 0.78, Math.sin (teta)* resorte.width * 0.81)*

Sin embargo, como se debe tomar la distancia real en pixeles, en las dos primeras líneas de código de la figura 43 se muestra la posición inicial del bloque, donde la variable $Unmetro$ es la que lleva a cabo la relación metros-pixeles.

```
bloque = game.add.sprite(distancia_comienzo_rampa_x+Math.cos(teta)*resorte.width*0.78,
punto_en_y_posicioninicial_resorte-Math.sin(teta)*resorte.width*0.81, 'bloque')
bloque.scale.setTo(game.width / 1100);
bloque.anchor.set(0,1);
bloque.rotation=-teta
bloque.animations.add('z', [0,1,2,4], 5, true)
```

Figura 43. Código para agregar el sprite de la figura 40.

Posteriormente se configura la secuencia animada que representa al bloque desplazándose dentro de la rampa y en movimiento parabólico, con base en los mismos procedimientos implementados en la práctica dos (movimiento en dos dimensiones), en la cual se empleó el método de interpolación lineal y el trazado de la trayectoria con las ecuaciones de posición (dentro del movimiento parabólico) respectivamente. Por medio de la animación denominada z se muestra la secuencia correspondiente al bloque siendo introducido en el blanco, cuando la distancia de compresión ingresada es la correcta.

2.8. PRÁCTICA 5: TRABAJO Y ENERGÍA

2.8.1. Diseño del sistema

La figura 44 muestra el sistema diseñado para la práctica de trabajo y energía en el cual un obrero dispone de tres cajas de igual masa e igual altura ($h_c = 40cm$) ubicadas en un punto inicial O y organizadas una encima de la otra. De acuerdo a la forma en que están acomodadas, cada caja tiene un color distintivo para diferenciar su posición respecto del suelo. De esta manera se tiene que la caja negra está a 0m del suelo, la caja azul a 0.40 m y la caja roja a 0.80 m.

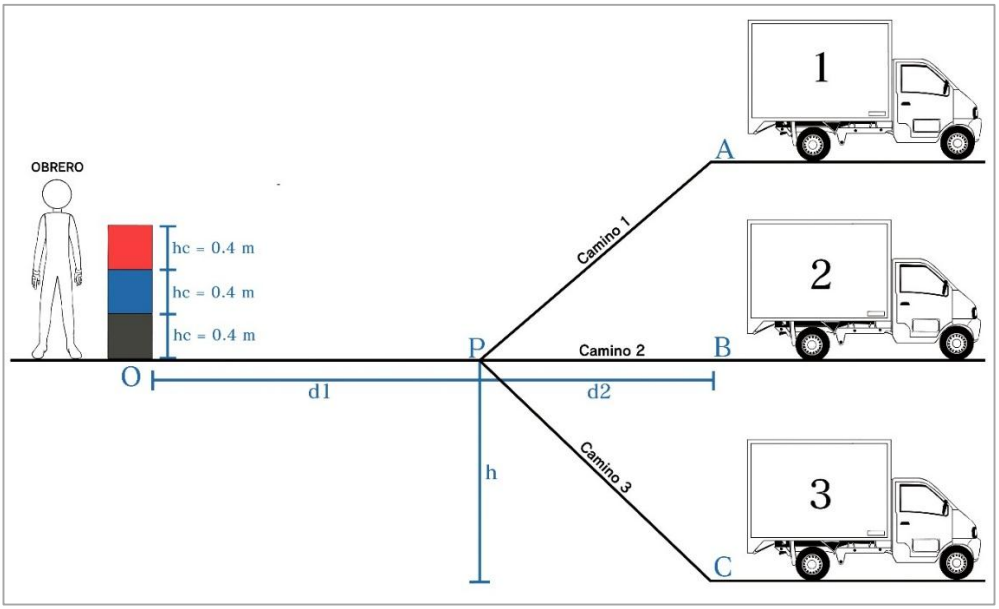

Figura 44. Práctica de trabajo y energía – Sistema general.

El funcionamiento del sistema permite que el obrero pueda escoger una de las cajas y transportarlas hasta cualquiera de los camiones ubicados al final de los caminos 1(tramo PA: plano inclinado ascendente), 2 (tramo PB: camino recto) y 3 (tramo PC: plano inclinado descendente). Al finalizar el recorrido se podrá determinar el trabajo total realizado por el obrero.

2.8.1.1. Objetivo de la práctica de trabajo y energía

Comprobar de manera experimental los valores de energía cinética y potencial de un cuerpo de acuerdo a las variaciones del trabajo realizado por las fuerzas que actúan dentro de un sistema dinámico definido por secciones. En esta comprobación se involucran una serie de cálculos complementarios en los que el

estudiante tiene la posibilidad de obtener matemáticamente diferentes valores de trabajo del obrero de acuerdo a la escogencia inicial de la caja en el punto de partida así como también los valores de velocidad al final de los camino 1 y 3 partiendo de las definiciones de energía cinética y potencial respectivamente, entre otros.

2.8.2. Diseño de la aplicación

El algoritmo diseñado para la práctica de trabajo y energía se muestra en la figura 45. Las variables de entrada que se manejan dentro de la simulación son las siguientes: θ_Rampa (ángulo de elevación de la rampa 1 – Camino A), h (altura de la rampa de la rampa 2 – Camino C), miu (coeficiente de fricción de los tres caminos), $masa$ (masa del bloque) y $Opción$ (corresponde a la opción de simulación que se escoge).

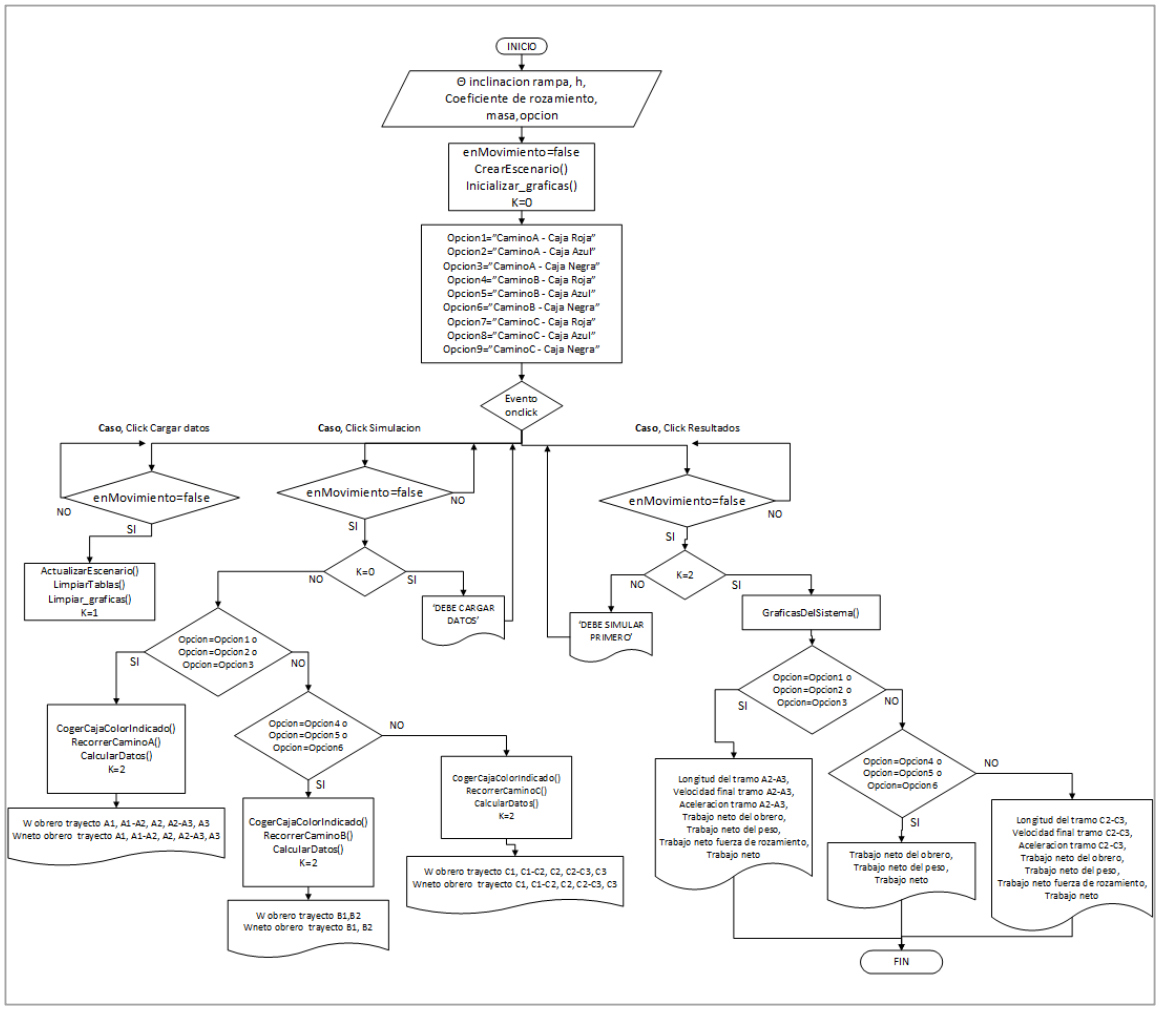

Figura 45. Algoritmo de la práctica de trabajo y energía.

Posteriormente se tiene proceso similar a los anteriores algoritmos en el que son llamadas las funciones encargadas de inicializar las gráficas y de crear el respectivo escenario de simulación. Dicho escenario está conformado por el personaje que representa al obrero, un grupo de cajas ordenadas verticalmente y tres caminos por los cuales se podrá desplazar dicho obrero cargando alguna de las cajas. También se tiene un proceso a través del cual se definen los nueve posibles casos de simulación, estos se basan en la escogencia del camino que se desea recorrer y la caja que se quiere llevar al final del mismo, de esta forma se incluyen las nueve opciones definidas en la descripción del sistema.

La simulación cuenta con los mismos botones de las prácticas pasadas (Cargar datos, Simulación y Resultados). En cuanto a la opción de simulación, el sistema pone a disposición las siguientes alternativas:

a) Que sea escogida la opción 1, la opción 2 o la opción 3, las cuales corresponden a la simulación del movimiento del obrero por el camino A.

b) Que sea seleccionada la opción 4, la opción 5 o la opción 6 (camino B).

c) Que sea escogida la opción 7, la opción 8 o la opción 9 (camino C).

Los datos de salida y las gráficas de movimiento dependerán de la opción que haya sido seleccionada. Al hacer click en el botón de simulación se llevan a cabo las diferentes secuencias de animación y además en algunos puntos referenciados de los tres caminos se irán mostrando de manera instantánea los valores de trabajo realizados por el obrero. Cuando es presionado el botón de resultados se mostrarán por medio de una tabla los resultados correspondientes a longitud del trayecto, velocidad de llegada de las cajas en los tramos inclinados, aceleración y valores de trabajo neto de las fuerzas que actúan sobre estas y trabajo del obrero.

2.8.2.1. Ecuaciones de movimiento

De acuerdo a la figura 44 se definen las expresiones generales que representan el trabajo neto (W_{Neto}) realizado por el obrero al levantar la caja negra y llevarla a través del camino 1 del sistema (plano inclinado ascendente):

$$W_F = -mgh = -mg(0.8) = 0.8mg \quad \textbf{(34)}$$

$$W_F = -mgh = -mg(0.8) = -0.8mg \quad \textbf{(35)}$$

$$W_F = \left(F_1 \cos\beta\right) L_1 \quad \textbf{(36)}$$

Donde L_1 y θ_1 corresponden a la longitud y al ángulo de inclinación de la rampa 1, m es la masa de la caja, μ_1 es el coeficiente de rozamiento de la superficie de la rampa, P_y y F_{1y} son las componentes en dirección y del peso y de la fuerza aplicada sobre la caja respectivamente y β es la dirección de aplicación de dicha fuerza.

Nota: La gráfica de trabajo del obrero a través del camino A se muestra en el capítulo 3. El desarrollo completo de las expresiones físicas de la práctica de trabajo y energía se encuentra disponible en el numeral 5 del anexo A (ecuaciones de movimiento de los sistemas físicos diseñados).

2.8.3. Proceso de codificación: práctica de trabajo y energía

Nota: Algunos métodos utilizados dentro de este apartado ya fueron usados en la codificación de la práctica 2, por lo tanto su uso solo será mencionados sin entrar en mayores detalles.

En primer lugar, considerando que son nueve los casos para escoger el tipo de simulación que se quiere realizar, se utiliza la etiqueta <select> para poder seleccionar una de las opciones disponibles. Con respecto a la creación del fondo del escenario de simulación se emplearon algunos métodos de Canvas para realizar la implementación de los tres trayectos que recorre el obrero, entre estos están:

a) En el primer trayecto (tramo recto y plano inclinado ascendente) se utilizaron los métodos **moveTo ()**, **lineTo ()**, **fillRect ()**, **arc ()**, entre otros, de manera similar a como se trató en la creación de las rampas del movimiento bidimensional.

b) En el segundo trayecto (tramo completamente recto) se empleó el método **fillRect ()** para realizar el bosquejo de un rectángulo que simulara la pista que conduce hacia el camión número 2.

c) La variante en cuanto al dibujo con Canvas se aprecia en el tercer trayecto, ya que se determinó que la superficie por donde debían desplazarse las cajas para llegar al camión 3 debía tener la forma de una pequeña meseta, con su correspondiente tramo recto y su caída inclinada, como se muestra en la figura 46.

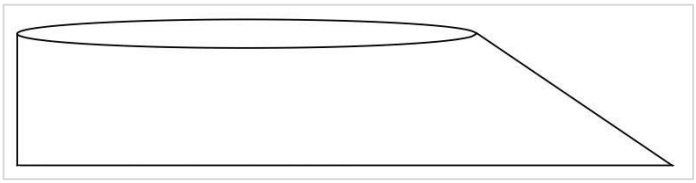

Figura 46. Imagen del trayecto tres la práctica de trabajo y energía.

Para realizar este dibujo a través de Canvas se empleó el método **ellipse (x, y, radioX, radioY, rotación, anguloinicio, angulofinal)** junto con otros de los métodos disponibles para formar una figura que permitiera en este caso la variación de la altura, la cual forma parte de los datos de entrada del sistema.

Por otra parte, cuando el fondo ha sido creado, se superpone la imagen del estante contenedor de las cajas con uno de sus espacios vacíos de acuerdo a la caja que ha sido escogida para ser transportada. Sobre esta misma posición se agrega el sprite de los movimientos iniciales del personaje en los que realiza la escogencia de dicha caja. De esta manera, cuando el personaje inicia una nueva secuencia de movimientos (desplazamiento con la caja escogida), la imagen del estante antes mencionada queda visible en la posición en que fue ubicada. El sprite de este elemento contenedor de las cajas se muestra en la figura 48:

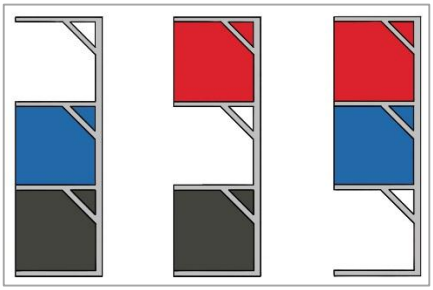

Figura 47. Imágenes de los tres estantes.

En cuanto a los sprites para las animaciones del movimiento del obrero se emplearon tres tipos, correspondientes a las secuencias que ilustran el transporte de las tres cajas desde el punto de partida hasta cada uno de los camiones disponibles al final de cada camino. Por ejemplo, el sprite que simboliza la secuencia del movimiento del obrero con la caja gris se muestra en la figura 48.

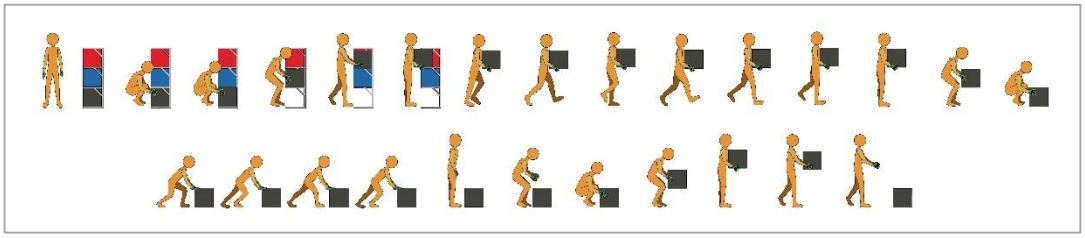

Figura 48. Sprite para realizar la secuencia de movimientos con la caja gris.

En la figura 49 se muestra la estructura del código mediante la cual se agregó el sprite al escenario de simulación y la manera en que se configuraron las respectivas secuencias con base en el grupo de imágenes contenidas en dicho sprite. La opción de simulación escogida corresponde al camino A, en donde se observa que:

```
else if(opcion=="Camino A - Caja Negra"){

personaje = game.add.sprite(punto_inicial_x_estanteYsprite, punto_inicial_y_estanteYsprite_caminoA, 'personaje1')
personaje.scale.setTo(game.width / 1000);
personaje.anchor.set(0,1);
personaje.animations.add('dos', [6, 7, 8, 9, 10], 4, true)
personaje.animations.add('uno', [0, 1, 2, 3, 4, 5], 1.7, true)
personaje.animations.add('tres', [12, 13, 14], 1.9, true)
personaje.animations.add('cuatro', [15, 16, 17, 18], 3, true)
personaje.animations.add('cinco', [19, 20, 21, 22, 23], 1.9, true)
personaje.animations.add('seis', [24,25], 5, true)

}
```

Figura 49. Código para agregar el sprite de la figura 48 en el camino A.

En la estructura se llevan a cabo los siguientes procesos:

a) La posición inicial del escenario en que se agrega el sprite de imágenes es la misma donde se ubica el sprite del estante de la figura 47.

b) De acuerdo a la figura 49 se tienen seis secuencias de animación dentro del sprite las cuales están representadas por las siguientes imágenes:

Animación 1: El obrero escoge alguna de las cajas (imágenes 1, 2, 3, 4 y 5).

Animación 2: El obrero realiza su desplazamiento por el tramo recto del camino A (imágenes 6, 7, 8, 9 y 10).

Animación 3: El obrero descarga la caja al final del tramo recto, antes del plano inclinado (imágenes 12, 13 y 14).

Animación 4: El obrero empuja la caja por la rampa inclinada (imágenes 15, 16, 17 y 18).

Animación 5: El obrero recoge la caja al final del tramo inclinado y la levanta para depositarla en el camión (imágenes 19, 20, 21, 22 y 23).

Animación 6: El personaje deposita la caja en el camión 1 (imágenes 24 y 25).

Una vez que se han definido las animaciones y la respectiva lógica para agregar los sprites se emplea el método de interpolación lineal para determinar los puntos de la trayectoria del movimiento del obrero. Este proceso se llevó a cabo dentro de la función **Simulacion ()**, de forma similar a como se implementó en la práctica de vectores.

Por último, dado que en la práctica se incluye una tabla donde se va mostrando en tiempo real los datos del trabajo (W) que va realizando el obrero en los puntos referenciados del trayecto, es necesario enviar el dato al respectivo elemento de la tabla dentro de la función Update mediante **document.getElementByID('')** ya que es a través de esta función que se asigna a la posición del sprite los puntos calculados de la trayectoria para que sean recorridos.

2.9. PRÁCTICA 6: COLISIONES

2.9.1. Diseño del sistema

El sistema general diseñado para la práctica de colisiones consiste en un juego de billar pool el cual cuenta con dos bolas de billar (bola 1 y bola 2), un taco y dos buchacas (buchaca 1 y buchaca 2) habilitadas en dos de esquinas de la mesa, como se muestra en la figura 51. Dentro de la mesa de billar se ha dispuesto un sistema de ejes coordenados que permiten referenciar la ubicación de las bolas y de las buchacas a través de puntos contenidos en el plano (x, y).

De esta manera la bola 1 (bola de impacto) se ubica en un punto fijo P_1 que demarca la posición de su centro mientras que las buchacas 1 y 2 se ubican en los puntos P_{b1} y P_{b2} respectivamente. Por su parte la bola 2 (bola objetivo) se podrá posicionar en cualquier punto P_2 dentro de un área específica de la mesa denominada "área delimitada para la ubicación de la bola 2".

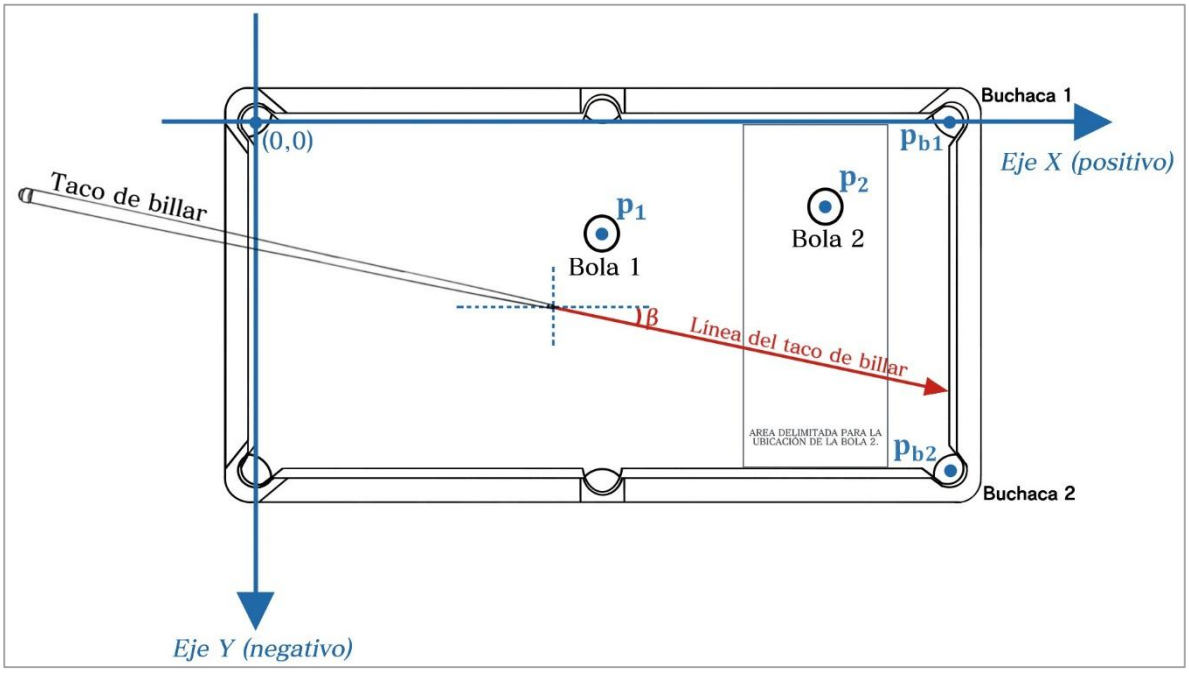

Figura 50. Práctica de colisiones – Sistema general.

Con base en las coordenadas suministradas (P_1, P_2, P_{b1} y P_{b2}) dentro del plano cartesiano y el valor del radio de las bolas de billar ($R_1 = R_2 = 3.52\text{cm}$) el usuario deberá realizar los cálculos matemáticos correspondientes que le permitan encontrar el valor del ángulo β (dirección del taco de billar) y la fuerza del taco $\mathrm{F_t}$ necesaria para ingresar la bola 2 de forma directa en alguna de las buchacas. Los resultados obtenidos se deberán corroborar a través de la herramienta de simulación.

2.9.1.1. Objetivo de la práctica de colisiones

Comprobar de manera experimental los conceptos de impulso y conservación del momento lineal dentro de un sistema que representa la colisión de dos objetos con igual masa. Con base en lo anterior, la práctica brinda al estudiante la posibilidad de trabajar directamente los conceptos de impulso y cantidad de momento lineal de un cuerpo, velocidades de los cuerpos antes y después de una colisión, direcciones de movimiento de los cuerpos después de una colisión, entre otros.

2.9.2. Diseño de la aplicación

El algoritmo de la práctica correspondiente a la temática de colisiones presenta algunos cambios en su estructura respecto a los que ya fueron estudiados.

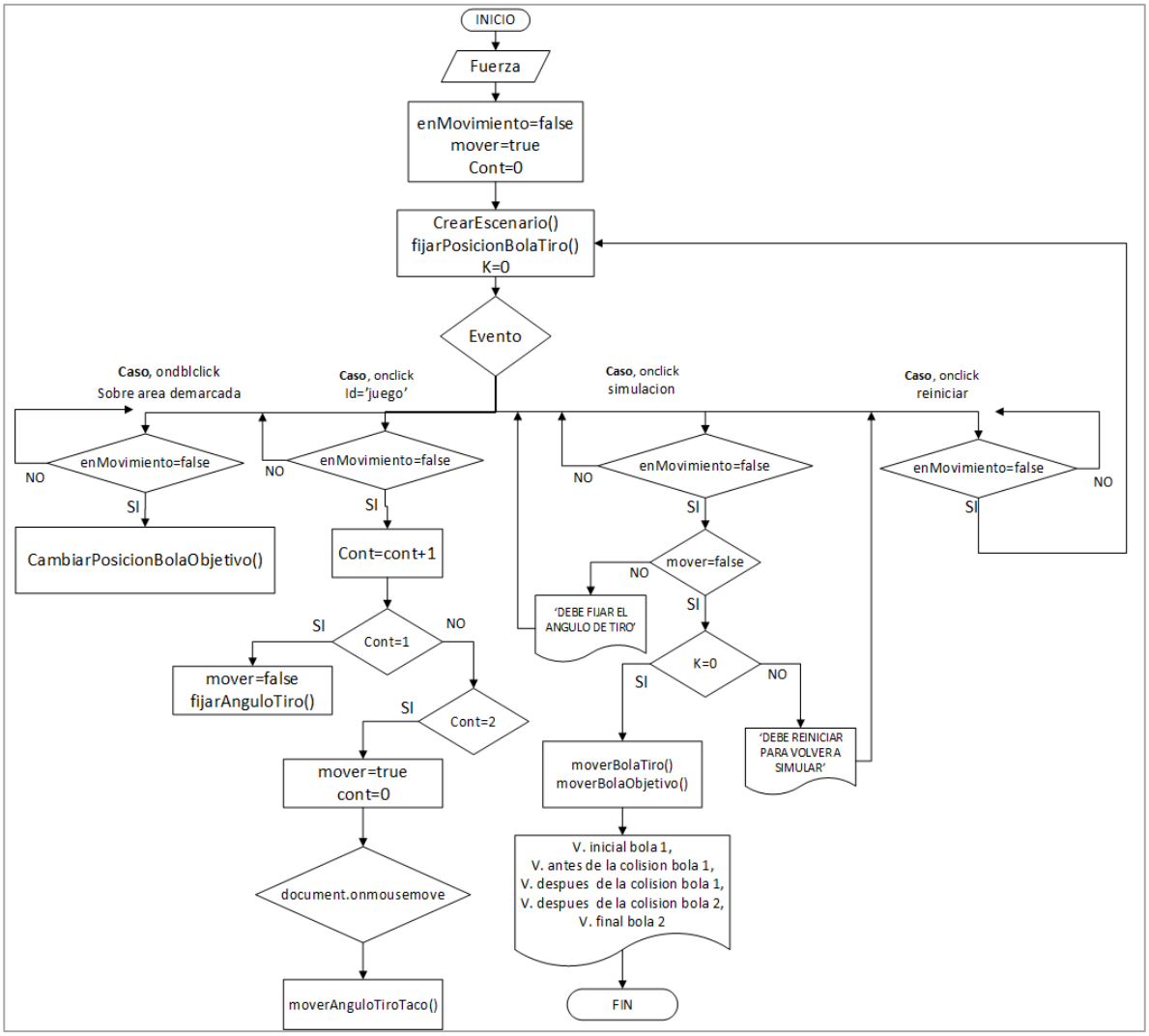

Figura 51. Algoritmo de la práctica de colisiones.

De acuerdo a la figura 51 el sistema inicialmente recibe como dato de entrada la fuerza con la cual el taco de billar golpea a la bola de tiro. Seguidamente se cuenta con un proceso en el cual se inician algunas variables de control definidas de la siguiente manera:

- **enMovimiento:** Variable utilizada para indicar el momento en que las bolas de la mesa están en movimiento. Es inicializada con el valor de *false.*

- **Mover:** Es la encarga de determinar si el ángulo de tiro del taco de billar va a depender del movimiento dado por el mouse, de esta manera, cuando la variable toma el valor *true* el ángulo de tiro varía según la posición del mouse dentro de la mesa y cuando toma el valor *false* el ángulo de tiro queda fijo.

- **Cont:** Es una variable que se encarga de llevar el control del número de veces que se hace click sobre la mesa de billar, ya que después de haber sido fijada la bola 2 se requiere un click para fijar el ángulo de tiro y otro click para volverlo a mover. Esta variable se inicializa con un valor de “0”.

Seguidamente se tiene otro proceso, utilizado para llamar a las funciones encargadas de crear el escenario y de fijar la posición de la bola de tiro. También se dispone de la variable k con la cual se controla que la acción de simular siempre sea realizada después de hacer click en el botón reiniciar. También se tiene un condicional utilizado para evaluar los eventos de JavaScript que se pueden llegar a presentar (*onclick* y *ondbclick*). Dichos eventos solo tienen efecto siempre y cuando la variable $enMovimiento = false$. Los casos en que están presentes dichos eventos son los siguientes:

A. **Caso 1: Ondbclick sobre el área demarcada dentro de la mesa de billar:** Con este evento se permite cambiar la posición de la bola que será colisionada (bola 2), simplemente haciendo doble click sobre el área definida dentro de la mesa de billar. La posición de esta bola también puede ser cambiada ingresando las coordenadas de la posición en el panel de entrada y posteriormente haciendo click en el botón reiniciar.

B. **Caso 2: Onclick sobre el área demarcada dentro de la mesa de billar:** Cada vez que se hace click sobre esta área la variable $cont$ se va aumentando en una unidad, de esta forma, cuando adquiere el valor de “1” (primer click) el ángulo de tiro se fija y la variable $mover$ toma el valor de *false*, con lo cual el ángulo de tiro deja de depender del movimiento del mouse. Posteriormente al hacer otro click sobre esta área la variable $cont$ toma el valor de “2”, lo cual indica que se puede volver a variar el ángulo de tiro. En este punto la variable $mover$ toma el valor de *true* nuevamente y la variable $cont$ se vuelve a inicializar a “0”.

C. **Caso 3: Onclick sobre botón simulación:** Este botón solo tiene efecto si la variable $mover$ es igual a *false*, es decir, si primero se fija el ángulo de tiro y la variable $k = 0$ (el sistema sí fue reiniciado). Con esto se llaman a las funciones encargadas de definir las trayectorias que seguirán las dos bolas de billar después de colisionar. También cuando se hace click sobre este botón se muestran los datos de salida en una tabla.

D. **Caso 4: Onclick sobre botón reiniciar:** Con este botón se permite volver las bolas a sus posiciones iniciales para volver a simular.

2.9.2.1. Ecuaciones de movimiento

Las expresiones generales de las velocidades de la bola 1 (inicial y después de la colisión) y de la bola 2 (después del choque) respectivamente vienen dadas de acuerdo al sistema mostrado en la figura 51:

$$v_{1i} = \sqrt{\left(\frac{\Delta t F_t}{m_1}\right)^2 - 2\mu g x_0} \quad \textbf{(37)}$$

$$v_{1f} = v_{1i}\sin\theta \quad \textbf{(38)}$$

$$v_{2f} = v_{1i}\cos\theta \quad \textbf{(39)}$$

Donde Δt corresponde a la duración del impulso dado por el taco de billar sobre la bola 1, F_t es la fuerza utilizada por el taco, μ es el coeficiente de rozamiento de la superficie de la mesa de billar, x_0 es la distancia inicial de separación entre las bolas 1 y 2, y finalmente θ es la dirección con que se mueve la bola 2 después de la colisión.

Nota: El desarrollo completo de las expresiones físicas de la práctica de colisiones se encuentra disponible el numeral 6 del anexo A (ecuaciones de movimiento de los sistemas físicos diseñados).

2.9.3. Proceso de codificación: práctica de colisiones

La práctica de colisiones incluye dentro de su estructura la parte concerniente a la animación del juego de billar y el despliegue de ciertos resultados numéricos. Por su parte se omitió la implementación de algún tipo de gráficas de movimiento del sistema. Teniendo en cuenta lo anterior, los componentes más relevantes del desarrollo de la aplicación para la presente práctica y sus respectivos procedimientos son los siguientes:

En primer lugar el sistema cuenta con dos botones denominados “Reiniciar” y “Simulación” para llevar a cabo las acciones dentro de la simulación del juego de billar diseñado. Mediante el botón “Simulación” se activan los movimientos de los elementos incluidos en el sistema, mientras que con el botón “Reiniciar” se devuelven los elementos a su posición y estado inicial.

Este botón está asociado con la función **CargarDatos ()** en la cual se adicionan los sprites de movimiento y las imágenes complementarias de la interfaz de simulación. Algunas de estas imágenes se muestran en la figura 52, en ella se tienen la representación del taco de billar, la bola de impacto (bola blanca) y la bola objetivo (bola amarilla).

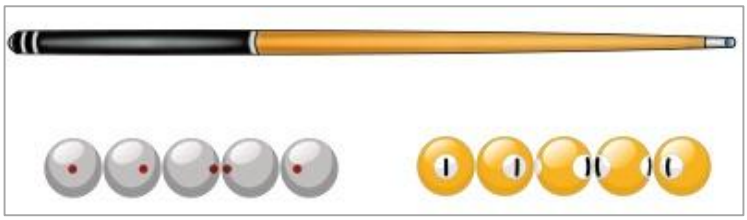

Figura 52. Elementos de la mesa de billar.

En la función **CargarDatos ()** se da la posición inicial de la bola blanca cuya coordenada va a estar fija para todas las simulaciones y se establece también la posición inicial de la bola amarilla, la cual puede ser cambiada a través de un doble click del mouse sobre el área definida para tal fin dentro de la mesa, o en su defecto ingresando las coordenadas en el panel de entrada respectivo y presionando el botón de reinicio de la página web.

Además de adicionar las imágenes en la función **CargarDatos ()** también se crean los BitmapData o elementos Canvas requeridos de manera separada para graficar y superponer trazos sobre los elementos de simulación, mientras se realiza la ubicación de la bola amarilla y la búsqueda del ángulo de tiro. Cuando se presiona el botón de simulación se borran estos bosquejos, correspondientes al ángulo barrido por el taco en busca de la dirección de tiro y la línea del taco que pasa por el centro de la bola de impacto (bola blanca). También se borra el recuadro que demarca el área de ubicación de la bola objetivo (bola amarilla).

Por otro lado, para llevar a cabo la simulación todo va a depender de la posición que tome el mouse a medida que es movido dentro del área de simulación, es decir, si su posición en y coincide con la posición en y de la bola amarilla se va a proyectar el área donde colisionará la bola blanca con la bola amarilla, de esta forma se irá mostrando el respectivo ángulo de tiro formado por el taco y un eje horizontal que pasa por el centro de la bola 1, el cual deberá ser mostrado en un campo visible dentro del panel de entrada de datos.

Partiendo del hecho de que el ángulo de tiro depende del movimiento del mouse sobre la bola amarilla, a medida que es buscado dicho ángulo se van mostrando gráficamente las direcciones que tomarían las bolas 1 y 2 después de la colisión, considerando que entre ellas forman un ángulo de 90°.

Retomando lo anteriormente dicho, dado que a través del mouse es que se determinan el ángulo de tiro y la ubicación de la bola 2, se requiere entonces conocer la posición del puntero, para ello se crearon las variables m y n, como se muestra en el código de la figura 53, las cuales toman la posición en x y y respectivamente.

```
m=Math.round(game.input.mousePointer.x*100)/100
n=Math.round(game.input.mousePointer.y*100)/100
```

Figura 53. Código para conocer la posición del mouse.

Como se requiere que estas variables se vayan actualizando cada vez que el mouse se mueva sobre el área de la mesa de billar, toda la lógica que las involucra se desarrolla dentro de la función **Update ()**. En cuanto al procedimiento para cambiar la posición de la bola 2 (bola objetivo) a través del doble click del mouse sobre el punto escogido, se empleó el evento ondbclick. La figura 54 muestra la estructura del código utilizada para esta funcionalidad:

```
document.getElementById("juego").ondblclick= function (ev) {

if((n>=punto_inicial_y_recuadroBlanco) && (n<=punto_final_y_recuadroBlanco) && (m>=punto_inicial_x_recuadroBlanco)
    &&(m<=punto_final_x_recuadroBlanco)){

if((!enMovimiento)) {
amarilla.x=m
amarilla.y=n
document.getElementById("x").value=Math.round(((m-punto_x_buchacaSuperiorIzquierda)/Uncm)*100)/100
document.getElementById("y").value=-Math.round(((n-punto_y_buchacaSuperior_izquierdaYderecha)/Uncm)*100)/100
}
else{return}
}
}
```

Figura 54. Código para mover la bola a colisionar.

Al hacer el doble click, dentro del div con identificador “juego” se llama a la función encargada de cambiar la posición de la bola, de esta forma si las posiciones n y m se encuentran sobre cierta área de la mesa la posición en x y y de la respectiva bola cambia mediante las propiedades amarilla.x y amarilla.y respectivamente. Cuando se ha fijado la posición de la bola objetivo es necesario escoger el ángulo de tiro, el cual también depende del movimiento del mouse sobre la esta bola. La lógica empleada para realizar esta acción es similar al procedimiento matemático denominado “geometría de la colisión”, cuyo desarrollo se encuentra disponible en el apartado 6 del anexo de ecuaciones de movimiento.

Posteriormente cuando se ha definido el ángulo de tiro es necesario fijarlo para que quede consignado en el cuadro de datos iniciales de la interfaz como un valor de entrada del sistema. Para ello se utilizan las funcionalidades del mouse a través de

un click. Nuevamente se utiliza el evento onclick, como se aprecia en a figura 55, en donde por medio de la variable $cont$, inicializada en cero, se establece un valor de 1 cuando se realiza el respectivo click. De esta manera la variable $mover$ toma el valor false y la dirección de tiro del taco de billar deja de depender del movimiento del mouse, quedando fijo el valor del ángulo que se estaba midiendo al momento de realizar el click.

Si se realiza un nuevo click sobre el área de simulación la variable $cont$ toma un valor de 2 y la variable $mover$ cambia al estado true, indicando que nuevamente la dirección de tiro depende del movimiento del mouse. Cuando esto ocurre la variable $cont$ vuelve a su valor inicial de cero. Dentro de esta función se establecen también los valores finales que se van a tomar para realizar los respectivos movimientos por medio de la simulación.

```
document.getElementById("juego").onclick= function (ev) {
    cont=cont+1

    if(cont==1){
    mover=false

   if(cont==2){
       cont=0
       mover=true }
```

Figura 55. Función para fijar el ángulo de tiro.

Con el valor del ángulo de tiro calculado se realiza el proceso de simulación, sin embargo se deben encontrar los puntos de las bandas en que las bolas pueden llegar a golpear después de la colisión. Para ello se empleó la siguiente ecuación de la recta que pasa pos dos puntos:

$$\frac{x-x_1}{x_2-x_1}=\frac{y-y_1}{y_2-y_1} \tag{40}$$

Con lo cual se obtienen las ecuaciones respectivas para x y y:

$$x=x_1+\frac{(x_2-x_1)(y_2-y_1)}{y_2-y_1} \tag{41}$$

$$y=y_1+\frac{(y_2-y_1)(x-x_1)}{x_2-x_1} \tag{42}$$

De esta manera, para determinar la dirección de la bola 2 se emplearon los puntos p_1 y p_2 y los puntos p_1 y p_3 para la dirección de la bola 1, como se observa en la figura 56.

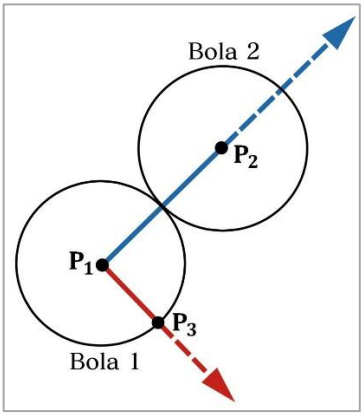

Figura 56. Direcciones que toman las bolas a colisionar.

Por su parte en la figura 57 se observa el procedimiento realizado dentro del respectivo código para calcular el punto al que llega la bola 2 después de la colisión, de acuerdo al caso mostrado en la figura 56, en la cual se observa lo siguiente:

```
xmes2=w1+((punto_y_buchacaInferior_izquierdaYderecha-u1)*(amarilla.x-w1))/(amarilla.y-u1)
    if(xmes2>punto_x_buchacas1y2){
        ymes2=u1+((punto_x_buchacas1y2-w1)*(amarilla.y-u1))/(amarilla.x-w1)

        xmes3=punto_x_buchacas1y2-Math.cos(teta_amarilla1)*amarilla.width/2
        ymes3=ymes2-Math.sin(teta_amarilla1)*amarilla.width/2

    }
    else{

        xmes3=xmes2-Math.cos(teta_amarilla1)*amarilla.width/2
        ymes3=punto_y_buchacaInferior_izquierdaYderecha-Math.sin(teta_amarilla1)*amarilla.width/2
    }
```

Figura 57. Código para determinar el punto de las bandas al cual llega la bola 2.

a) Como se conocen las posiciones de las bandas en pixeles primero se determina la coordenada x fijando a y en un valor de 184 pixeles que corresponde a la posición en y de la banda superior. Dentro de este procedimiento la coordenada x recibe el nombre de $xmes2$, en la cual el punto 1 corresponde a las coordenadas $(w1, u1)$ y el punto 2 corresponde a $(amarilla.x, amarilla.y)$.

b) Si el valor de x hallado es mayor que 863 (que corresponde a la coordenada en x de la banda derecha) significa que la bola no pega en la banda superior

si no en la banda derecha. De esta forma se fija el valor de x en 863 pixeles y se procede a encontrar el valor de y alcanzado por la bola por medio de la variable que se ha denominado $ymes2$.

c) Las variables $xmes3$ y $ymes3$ corresponden a los puntos finales a los cuales llegará la bola 2. Estos son una adaptación de los puntos $xmes2$ y $ymes2$ con los cuales se hará que la bola 2 toque el punto no con su centro (como está referenciado en las imágenes del sprite) si no con algún punto contenido en el perímetro de la misma, de modo que se represente correctamente el toque de la bola con las respectivas bandas. Si se quiere determinar el punto en donde golpea la bola de impacto (bola blanca) se recurre al mismo análisis realizado para este caso.

Esta explicación corresponde solo a uno de los casos en los que se hallan los puntos de choque de las bolas con las bandas después de que ha ocurrido la colisión. De esta forma, al haber sido establecidos dichos puntos se hizo uso del método de interpolación lineal para encontrar el resto de puntos de las trayectorias lineales que recorrerán las bolas dentro de la mesa.

Finalmente se determinan los rebotes de las bolas con las respectivas bandas de la mesa, considerando que dicho rebote depende del ángulo de incidencia con que estas bolas golpeen las bandas, como se aprecia en la figura 58.

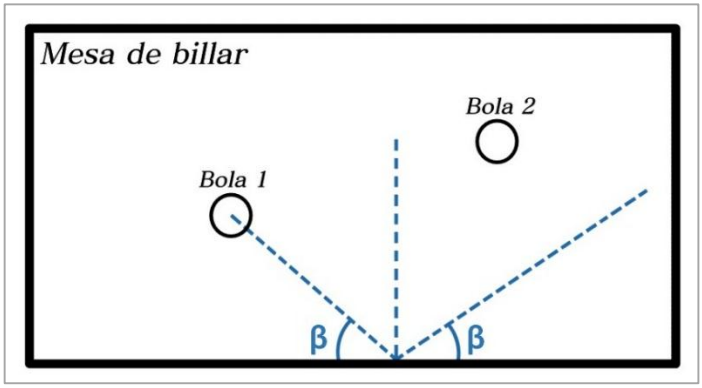

Figura 58. Lógica para hallar los rebotes sobre las bandas.

Capítulo 3

RESULTADOS

En este capítulo se muestran los resultados del proceso de diseño y codificación de las seis prácticas establecidas inicialmente y se demuestra, a partir de casos particulares de simulación, el correcto funcionamiento del producto software implementado. Además se dan a conocer los resultados de la respectiva evaluación realizada con los estudiantes que llevaron a cabo la prueba de la aplicación.

3.1. INTEGRACIÓN DE LOS SUBSISTEMAS

Con base en los sistemas físicos planteados para cada una de las prácticas definidas dentro del capítulo 2 se realizó la implementación de las seis aplicaciones independientes, cada una con sus características y propiedades particulares, las cuales han sido integradas dentro de una aplicación web, como se muestra en la figura 59.

Figura 59. Interfaz principal de la aplicación.

Como se observa, la interfaz principal de la aplicación exhibe una lista con las seis prácticas implementadas, el link respectivo para el ingreso a cada simulación y los documentos de apoyo para la realización de dichas prácticas (guías de actividades con los pasos para la realización de las simulaciones y el anexo de las ecuaciones de movimiento definidas a partir del análisis físico de cada sistema). La interfaz cuenta también con un elemento que permite mostrar los sistemas desarrollados a través de un deslizador de imágenes con efectos de transiciones implementado a través de WowSlider [75]. La aplicación se encuentra disponible en el siguiente enlace: http://www.unicauca.edu.co/experimentos_mecanica/aplicacion.html.

A continuación se exploran los resultados obtenidos luego del proceso de codificación e implementación de cada una de las seis simulaciones que componen el sistema total y se realiza una comparativa entre las gráficas de movimiento obtenidas a través de la aplicación con las gráficas realizadas a partir del software MatLab. Estas curvas están basadas en las ecuaciones de movimiento dadas en el capítulo 2.

3.2. PRUEBAS DEL SISTEMA

3.2.1. Práctica 1 - Vectores:

Con base en el sistema dado en el capítulo 2, las aplicaciones individuales que representan las actividades 1 y 2 de la práctica de vectores son las siguientes.

3.2.1.1. Actividad 1.

La figura 60 muestra la interfaz principal de la herramienta de simulación desarrollada para la actividad 1 de la práctica de vectores (representación de vectores en coordenadas polares). De acuerdo a las pautas y características dadas en la definición de la práctica, la interfaz contiene dos campos principales en los cuales se lleva a cabo la implementación gráfica de los vectores dentro del plano cartesiano y el ingreso de los datos iniciales junto con la visualización de los resultados numéricos respectivamente.

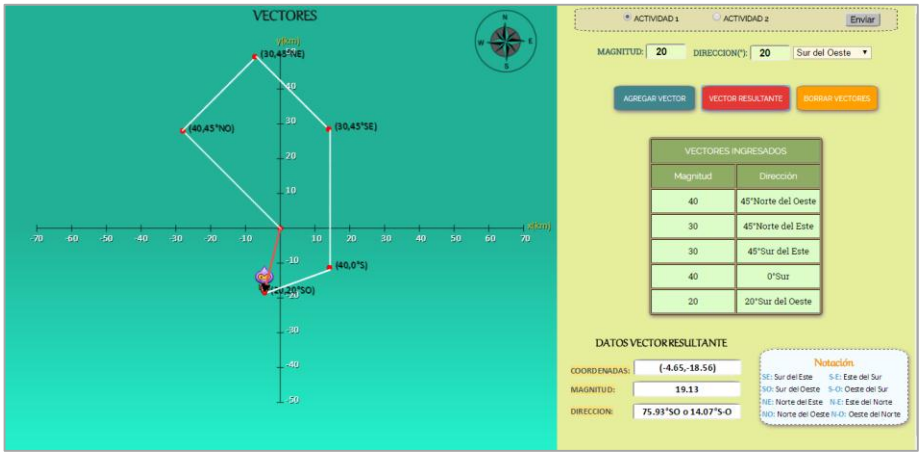

Figura 60. Interfaz de la actividad 1 de vectores.

De forma más explícita, el campo derecho de la interfaz contiene los botones respectivos para realizar los procedimientos de agregar un vector (por medio de su magnitud, dirección y orientación), de calcular el vector resultante y de borrar los vectores que hayan sido ingresados inicialmente. Esta sección también incluye los cuadros de captura de los datos de entrada, la tabla de vectores que han sido dibujados y los cuadros de despliegue de resultados numéricos (coordenadas, magnitud y dirección del vector resultante). Para la muestra, en la figura 60 se puede apreciar la construcción gráfica de la resultante de cinco vectores ingresados por el usuario.

3.2.1.2. Actividad 2.

Por su parte la actividad 2 de la práctica de vectores (representación de vectores en coordenadas cartesianas), cuya interfaz se muestra en la figura 61, en esencia contiene los mismos componentes de la aplicación desarrollada para la actividad 1, solo que para este caso se ha adicionado un campo extra para el ingreso del ángulo de rotación de los ejes coordenados, un botón por medio del cual se lleva a cabo el procedimiento de rotación y una tabla para mostrar las coordenadas de los segmentos orientados respecto a los ejes del plano rotados.

En la misma figura se observa de nuevo la construcción gráfica de un conjunto de cinco vectores dispuestos en diferentes cuadrantes del plano cartesiano y el correspondiente vector resultante. Además se muestra la rotación realizada sobre los ejes coordenados equivalente a 45° en sentido antihorario.

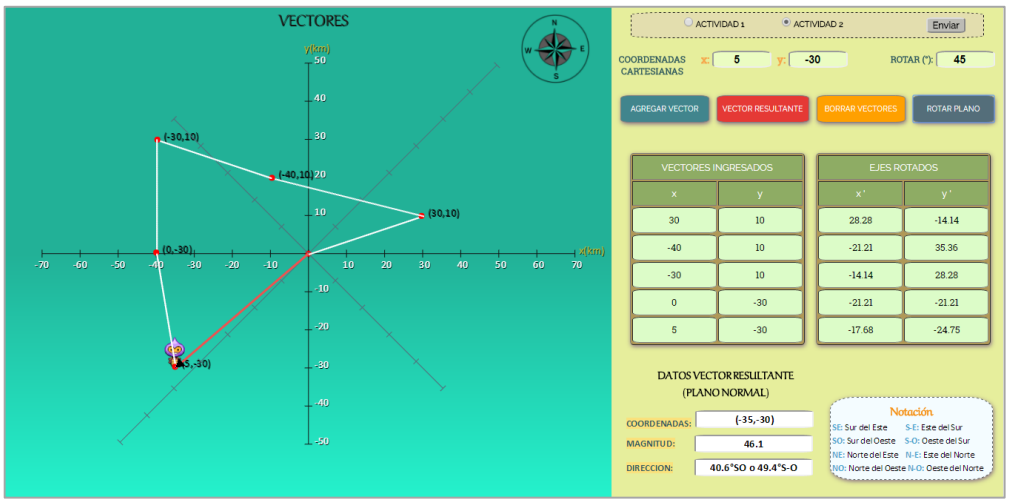

Figura 61. Interfaz de la actividad 2 de vectores.

3.2.2. Práctica 2 – Movimiento en dos dimensiones.

La implementación de la herramienta de simulación para el sistema físico de movimiento bidimensional diseñado se muestra en la figura 62. Similar a la práctica de vectores, la sección superior izquierda de la interfaz corresponde al área destinada para la animación del movimiento del objeto a través de los diferentes puntos del tramo total. En la sección superior derecha de la interfaz se encuentran los campos para el ingreso de los datos de entrada de la aplicación, el cuadro de opciones de simulación y los botones para la realizar los procedimientos de carga de dichos datos iniciales, inicio de simulación y despliegue de resultados.

Adicionalmente la aplicación de movimiento bidimensional cuenta con una sección en la que se muestran los resultados gráficos de cada simulación por medio de curvas de movimiento que ilustran el paso de la pelota por cada tramo del sistema. Las gráficas en cuestión corresponden a: posición en x vs tiempo (t), posición en y vs tiempo (t), velocidad (v) vs tiempo (t) y aceleración total (a) vs tiempo (t). Finalmente en la interfaz se muestran una serie de resultados numéricos derivados de la simulación y un cuadro sobre la notación de las variables utilizadas.

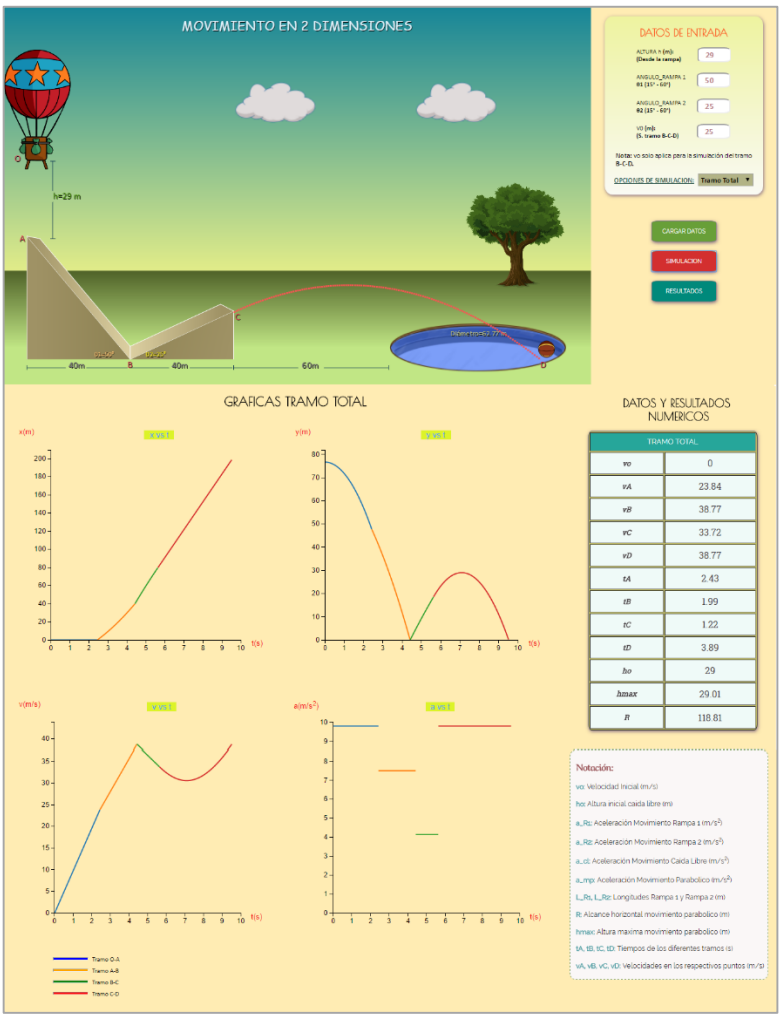

Figura 62. Interfaz de la aplicación de movimiento en 2D.

Para comprobar el correcto funcionamiento del sistema implementado se realiza la prueba de un caso particular con base en unos valores iniciales de altura de ubicación de la pelota (h), ángulo de elevación de la rampa 1 (θ_1) y ángulo de elevación de la rampa 2 (θ_2), los cuales son consignados en la tabla 9. Con base en estos datos de entrada se realiza la respectiva simulación del tramo total del sistema y se comparan los resultados gráficos obtenidos por medio de la herramienta MatLab y a través de la aplicación desarrollada.

Nota: Los resultados numéricos de la simulación se muestran en el cuadro respectivo dentro de la figura 62.

Tabla 9. Datos de entrada para la simulación – Movimiento en dos dimensiones.

h(m)	θ_1(°)	θ_2(°)
29	50°	25°

Una vez realizada la simulación se obtuvieron las respectivas gráficas de movimiento, de acuerdo a la figura 63.

Nota: En todas las gráficas de movimiento la imagen de la izquierda corresponde a la curva obtenida con la herramienta MatLab y la derecha es la que genera la aplicación.

- **Gráfica de posición en x vs tiempo (t):**

La figura 63 muestra la gráfica de posición en x vs tiempo (t) generada con las dos herramientas utilizadas para los cuatro tramos del sistema (tramos O-A, A-B, B-C y C-D) con base en las ecuaciones (18), (19), (20) y (21) respectivamente. De acuerdo a los resultados obtenidos se observa la concordancia entre los valores de desplazamiento en dirección x hallados en ambas curvas, los cuales vienen dados de la siguiente forma: En el movimiento de caída libre (de $t = 0\ s$ hasta $t = 2.43\ s$) se tiene que la pelota no efectúa ningún desplazamiento en x, ya que la traslación se lleva a cabo solamente en dirección y.

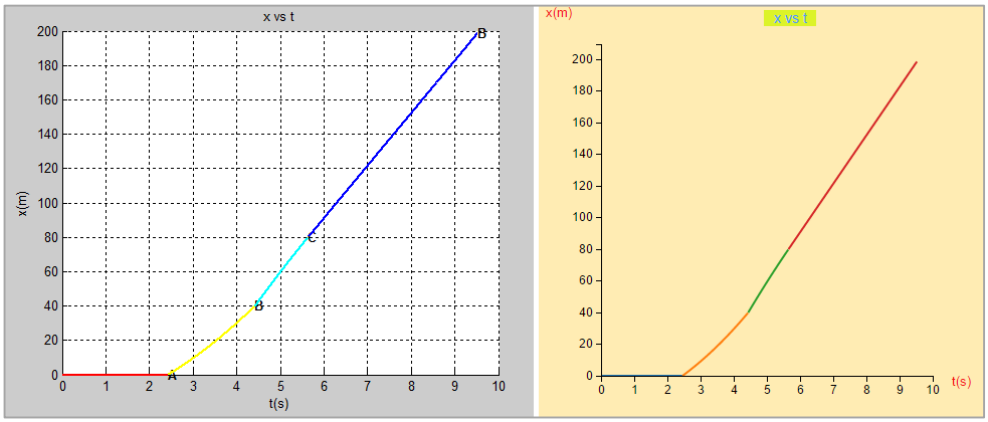

Figura 63. Graficas de x vs tiempo (t) – Movimiento bidimensional.

En cuanto a su paso por las rampas inclinadas, se evidencia que en ambos casos la pelota recorre los 40 metros que mide la base de cada una, en intervalos de tiempo de $t = 2.43\ s$ hasta $t = 4.42\ s$ para la rampa 1 y de $t = 4.42\ s$ hasta $t = 5.64\ s$ para la rampa 2. Esta diferencia de tiempos se da porque aunque el

desplazamiento está medido dentro del sistema de referencia total, el tiempo de recorrido si se considera con base al sistema de referencia individual establecido para cada rampa. Finalmente, en el tramo correspondiente al movimiento parabólico ($t = 5.64\ s$ hasta $t = 9.53\ s$) se muestra el alcance horizontal (R) de la pelota, el cual en ambos casos es de aproximadamente 118.81 metros.

- **Gráfica de posición en y vs tiempo (t):**

En la figura 64 se observan las curvas de posición en y vs tiempo (t) obtenidas con la herramienta MatLab y con el simulador respectivamente en los diferentes tramos del sistema, con base en las ecuaciones (22), (23), (24) y (25). Al igual que en el movimiento en x, las gráficas representan acordemente el desplazamiento de la pelota en dirección y en cada uno de los intervalos de tiempo definidos.

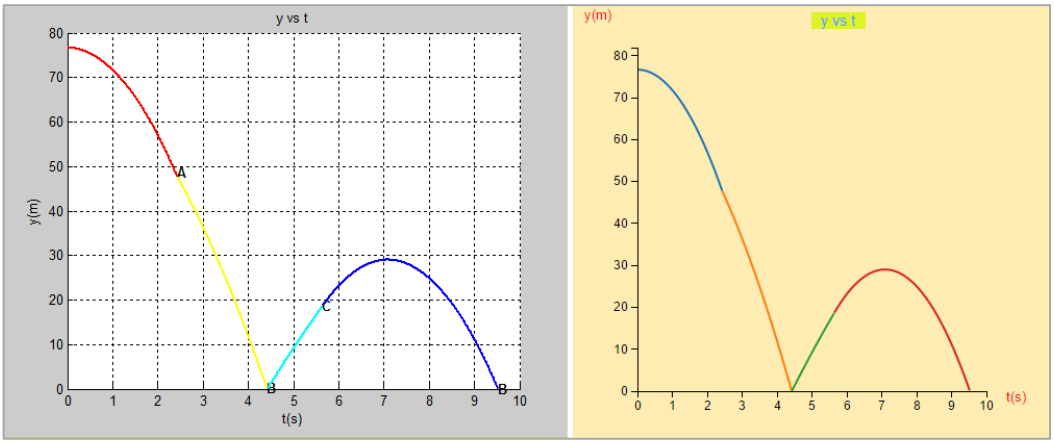

Figura 64. Graficas de y vs tiempo (t) – Movimiento bidimensional.

En el movimiento de caída libre se observa la altura que pierde la pelota en el tiempo t_A(tiempo en alcanzar la rampa 1), pasando de una posición inicial de $h = 76.67\ m$ hasta una altura de $h = 47.67\ m$. Por su parte, en su paso por la rampa 1 el balón en efecto desciende completamente hasta una altura $h = 0\ m$ en el intervalo de tiempo respectivo y logra ascender hasta una posición $h = 18.65\ m$ (altura de la rampa 2) a través del plano inclinado ascendente, punto desde el cual empieza su movimiento de tipo parabólico.

Las curvas muestran que después de partir del final de la rampa 2 la pelota alcanza la altura máxima $h_{máx} = 29.01\ m$ en el movimiento parabólico aproximadamente en $t = 1.4\ s$ y llega al punto más bajo en $t = 3.89\ s$. El tiempo de subida y de bajada del balón en este movimiento no es simétrico debido a que los puntos de inicio y de

final de dicho movimiento están ubicados a diferente altura, situación que se ve reflejada en la curva respectiva de la trayectoria parabólica.

- **Gráfica de velocidad (v) vs tiempo (t):**

Las curvas de velocidad del balón, mostradas en la figura 65 y que están construidas con base en las ecuaciones (14), (15), (16) y (17), presentan las siguientes características: dentro del movimiento de caída libre la pelota solo se mueve en dirección y, partiendo del reposo ($v = 0\ m/s$) desde el punto ubicado inicialmente. A medida que desciende, la pelota va adquiriendo una mayor velocidad gracias a que no hay oposición del medio en el que se mueve (aire) y tampoco se tiene en cuenta el concepto de velocidad terminal para objetos que caen bajo la acción de la gravedad, alcanzando un valor de $v = 23.84\ m/s$ en la punta de la rampa 1. Este valor de velocidad se sigue incrementando dentro del plano inclinado descendente, llegando a un máximo de $v = 38.77\ m/s$ al final del mismo, gracias en parte a que la superficie de este no presenta fricción.

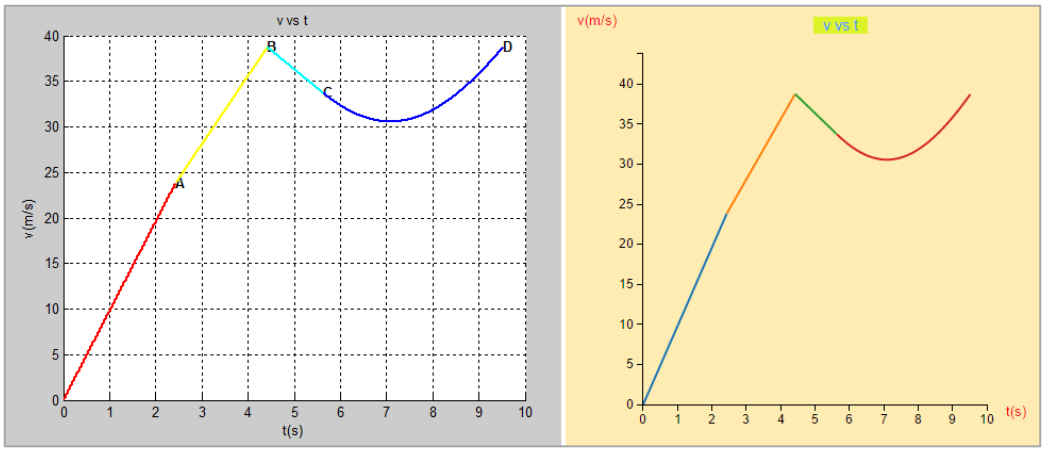

Figura 65. Gráficas de v vs tiempo (t) – Movimiento bidimensional.

Sin embargo, al llegar a la rampa 2 la dirección del movimiento cambia y el peso de la pelota actúa en su contra como consecuencia de que en este tramo el trayecto inclinado es de tipo ascendente, por tal motivo la velocidad del balón decrece hasta un valor de $v = 33.72\ m/s$. Finalmente en la trayectoria parabólica la pelota sigue perdiendo velocidad debido a la acción de la fuerza de gravedad que se opone al movimiento, hasta llegar al punto en que alcanza la altura máxima ($h_{máx} = 29.01\ m$) en donde su componente de velocidad en y decrece hasta el valor mínimo de $v_y = 0m/s$. A partir de este punto la dirección del vector de velocidad en y cambia y apunta en el mismo sentido que el vector de aceleración, de modo que la

velocidad empieza a incrementarse nuevamente hasta el valor de $v = 38.77\ m/s$, que es cuando el balón golpea el suelo.

- **Gráfica de velocidad (a) vs tiempo (t):**

Finalmente en la figura 66 se muestra los valores constantes de aceleración para cada intervalo de tiempo dentro del sistema total de movimiento en dos dimensiones. Nuevamente, las curvas generadas por las dos herramientas representan correctamente la magnitud de cada aceleración y la relación, de acuerdo al tipo de movimiento realizado, con los valores de los otros tramos, e ilustran por ejemplo, que la medida de la aceleración dentro de la rampa 2 concuerda con el decremento de velocidad del balón observado en el intervalo de tiempo de $t = 4.42\ s$ hasta $t = 5.64\ s$. También se observa que en el movimiento de caída libre y en el parabólico la pelota presenta la misma aceleración. Las expresiones utilizadas en la construcción de estas gráficas corresponden a las ecuaciones (11), (12), y (13).

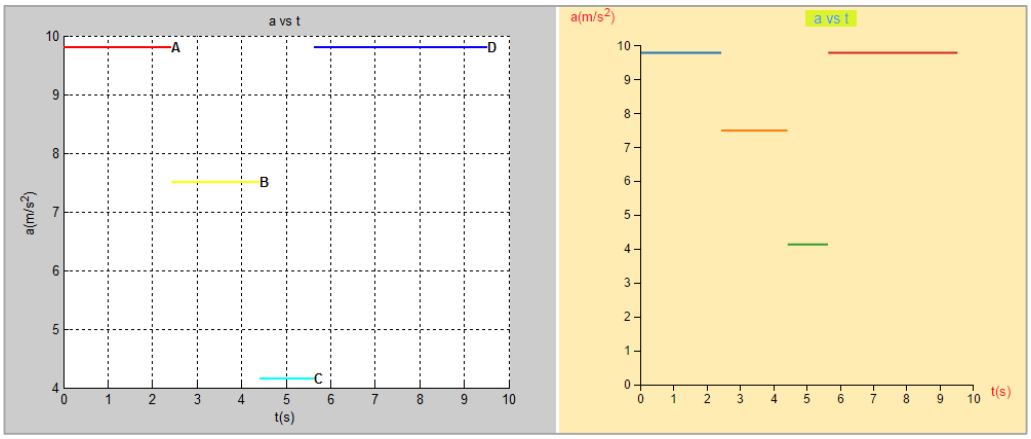

Figura 66. Gráficas de a vs tiempo (t) – Movimiento bidimensional.

3.2.3. Práctica 3 – Movimiento circular

La figura 67 muestra la interfaz de la aplicación desarrollada para la práctica de movimiento circular. Los botones respectivos y el campo para los datos de entrada conservan en esencia la misma funcionalidad. En cuanto a las curvas que ilustran el movimiento del motociclista en su paso por la pista de acrobacias y que son mostradas después de cada simulación, la aplicación realiza el despliegue de ocho tipos de gráficas correspondientes a: velocidad del piloto (v) vs tiempo (t), aceleración total del piloto (a) vs tiempo (t), velocidad del piloto (v) vs ángulo recorrido (θ), fuerza normal de la superficie del tramo circular (N) vs ángulo recorrido

(θ), aceleración tangencial del piloto (a_t) vs ángulo recorrido, aceleración radial del piloto (a_r) vs ángulo recorrido (θ) y aceleración total del piloto (a) vs ángulo recorrido (θ). Las gráficas de las variables realizadas respecto al ángulo recorrido (θ) corresponden exclusivamente al segmento circular.

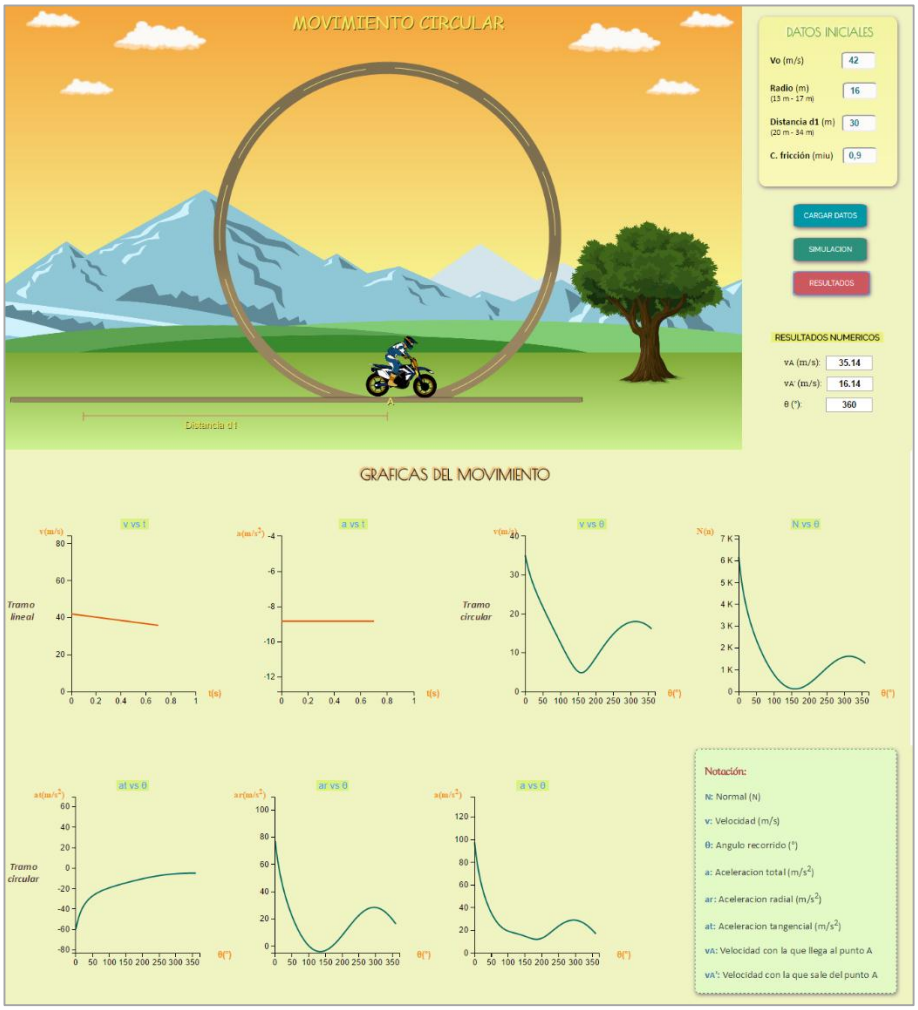

Figura 67. Interfaz de la aplicación de movimiento circular.

La comprobación del correcto funcionamiento del sistema implementado se realiza a través de una simulación con base en los datos de entrada, como se muestra en la tabla 2, correspondientes a velocidad inicial (v_0) con que parte el piloto en el segmento lineal, el radio de la plataforma circular (r), la distancia de separación entre el punto de partida del piloto y el inicio del tramo circular (d_1) y el coeficiente de fricción (μ) de la superficie total de la pista de acrobacias. Con estos valores se garantiza que el motociclista logra dar un giro completo a la circunferencia.

Tabla 10. Datos de entrada para la simulación – Movimiento circular.

v_0(m/s)	r(m)	d_1(m)	μ
42	16	30	0.9

Nota: Con el ánimo de no extender demasiado el presente documento, a partir de este punto solo se hará la revisión de una o máximo dos gráficas por cada práctica. También, las gráficas mostradas a continuación solo corresponden al movimiento circular del piloto.

- **Gráfica de velocidad del piloto (v) vs ángulo recorrido (θ):**

La figura 68 muestra la curva de velocidad del piloto respecto al ángulo recorrido en su paso por la plataforma circular, de acuerdo a la ecuación (26). El movimiento inicia con la misma velocidad con que el motociclista finaliza su desplazamiento dentro del tramo recto ($v = 35.14\ m/s$). A medida que recorre la circunferencia se observa como la velocidad va disminuyendo debido a las fuerzas que se oponen al movimiento, lo que provoca una desaceleración en la moto.

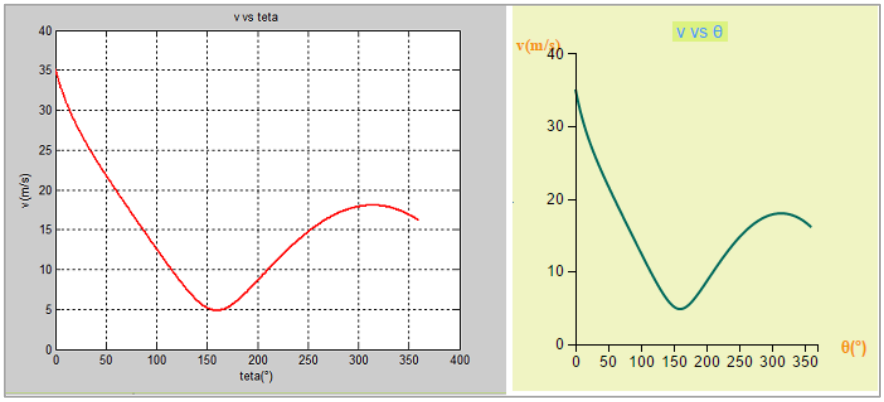

Figura 68. Gráficas de v vs θ – Movimiento circular.

Al llegar a un ángulo $\theta = 160°$ aproximadamente el piloto alcanza su valor crítico de velocidad, pero sin llegar a ser cero, lo que le permite seguir unido a la superficie de la plataforma. Es precisamente a partir de este punto en que empieza a incrementar la velocidad gracias a que empieza el tramo de descenso del movimiento circular. Al finalizar el giro el piloto ha recorrido los 360° de la circunferencia, alcanzando una velocidad de $v = 16.14\ m/s$.

- **Gráfica de aceleración total del piloto (a) vs ángulo recorrido (θ):**

Finalmente la figura 69 muestra las gráficas generadas para representar la aceleración total del motociclista, definidas a partir de las expresiones generales de aceleración tangencial (a_t) y aceleración radial (a_r). Dado que la velocidad del piloto cambia constantemente en magnitud y dirección dentro de este tramo, la aceleración no permanece constante y presenta el comportamiento mostrado en las curvas, cambiando de un punto a otro. Esta propiedad es la que define el movimiento circular uniformemente variado realizado por el piloto.

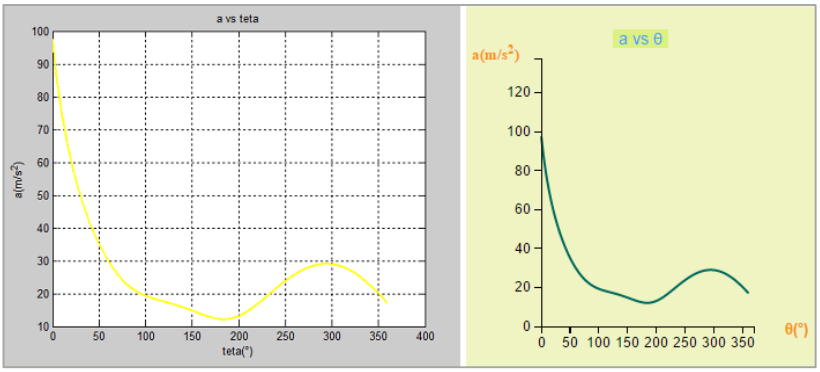

Figura 69. Gráficas de a vs θ – Movimiento circular.

3.2.4. Práctica 4 – Leyes del movimiento:

La interfaz de la aplicación desarrollada para la práctica referente a las leyes del movimiento se muestra en la figura 70. La novedad implementada dentro de esta aplicación es la posibilidad de obtener resultados numéricos en tiempo real, es decir, a medida que la simulación transcurre y el bloque va completando su recorrido se van mostrando, por medio de una tabla, algunos valores que representan su movimiento a través del sistema físico implementado.

La sección de gráficas del simulador incluye las siguientes curvas de movimiento: posición del bloque en dirección x vs tiempo (t), posición del bloque en dirección y vs tiempo (t), velocidad del bloque (v) vs tiempo (t) y aceleración del bloque (a) vs tiempo (t). En cuanto a la simulación realizada para la obtención de las respectivas curvas, se introdujeron los datos de entrada correspondientes a: longitud de compresión del resorte ($X_{Resorte}$), distancia al blanco (R), coeficiente de fricción de la rampa (μ) y ángulo de elevación de la rampa (θ). Los valores de entrada para llevar a cabo la comprobación del correcto funcionamiento de la aplicación están consignados en la tabla 11:

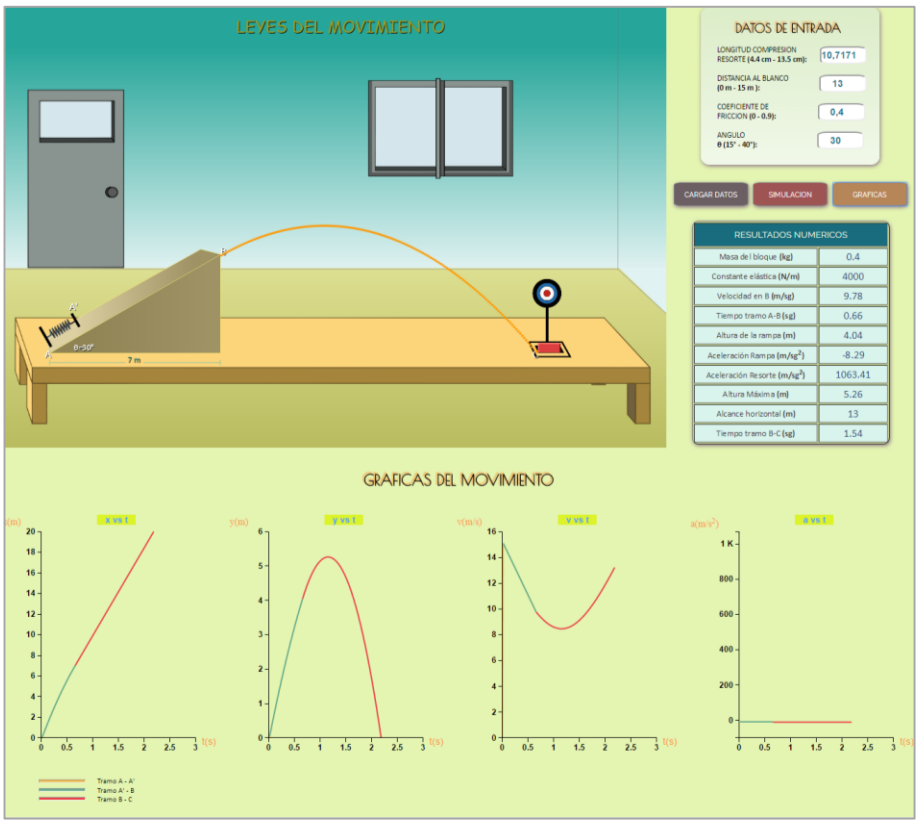

Figura 70. Interfaz de la aplicación de movimiento en 2D.

Tabla 11. Datos de entrada para la simulación – Leyes del movimiento.

$X_{Resorte}$(Cm)	R(m)	μ	θ(°)
10.7171	13	0.4	30

A continuación se muestran las curvas de velocidad para cada uno de los tramos generadas con la aplicación:

- **Gráfica de velocidad total (v) vs tiempo (t):**

La figura 71 muestra las gráficas de velocidad total del bloque en función del tiempo, basadas en las ecuaciones (30), (31) y (32). Se observa que en efecto el bloque alcanza un valor grande de velocidad en un instante muy corto de tiempo debido a la gran aceleración que en conjunto desarrolla el sistema masa-resorte, llegando a un valor aproximado de $v = 15\ m/s$ al finalizar el tramo A-A' (punto de equilibrio del muelle). El análisis de velocidad del bloque a lo largo de la rampa y en el movimiento parabólico es similar al de la gráfica respectiva dentro del movimiento en dos dimensiones.

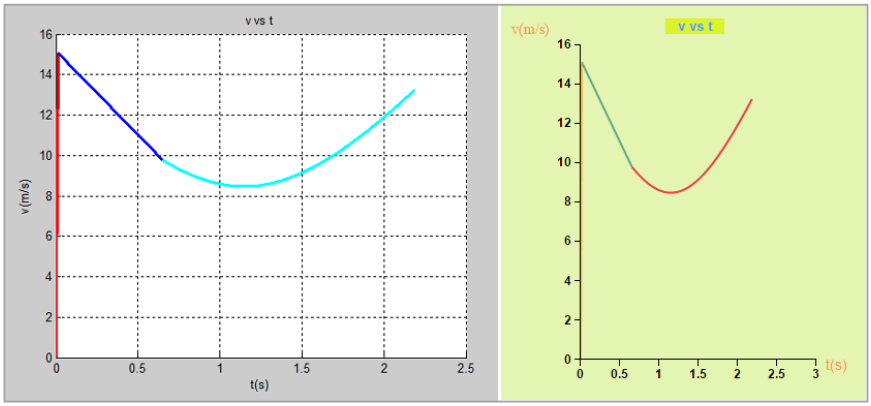

Figura 71. Gráficas de a vs t – Leyes del movimiento.

3.2.5. Práctica 5 - Trabajo y energía:

La práctica correspondiente al tema de trabajo y energía se muestra en la figura 72. Las curvas de movimiento incluidas dentro de la aplicación son las siguientes: gráfica de trayectoria de las cajas (x vs y), gráfica de variación de energía (x vs E) y gráfica de trabajo del obrero (x vs W). Los datos de entrada para simular uno de los casos disponibles dentro de la herramienta corresponde a: ángulo de inclinación de la rampa 1 (θ_1), altura de la rampa 2 (h), coeficiente de rozamiento (μ), masa de las cajas (m) y la opción de simulación (Opción).

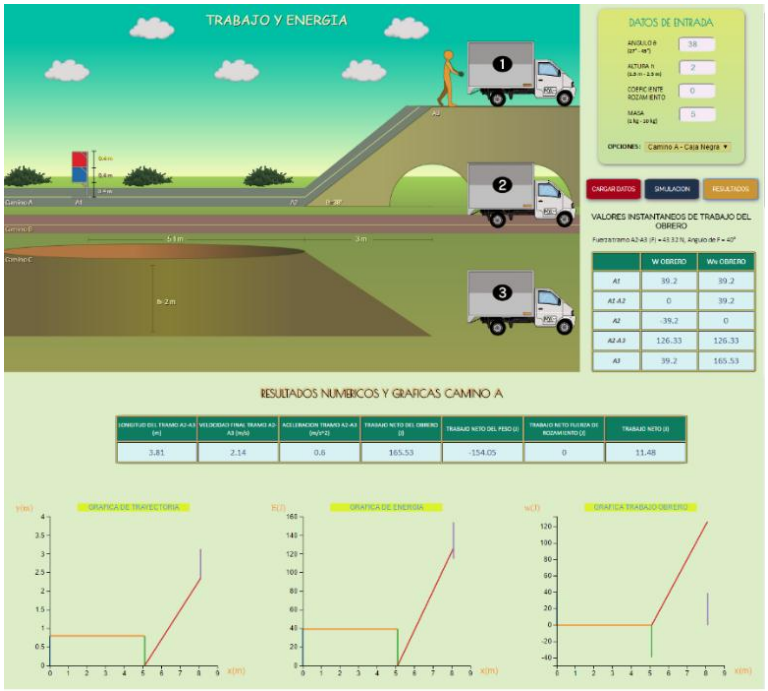

Figura 72. Interfaz de la aplicación de movimiento en 2D.

Los valores de los datos de entrada se muestran en la tabla 12:

θ_1(°)	h(m)	μ	m(Kg)	Opción
38	2	0	5	Camino A – Caja Negra

Tabla 12. Datos de entrada para la simulación – Trabajo y energía.

- **Gráfica de trayectoria x vs y:**

La gráfica de trayectoria (x vs y) mostrada en la figura 73 representa el recorrido horizontal y vertical que realiza la caja desde el punto de partida hasta que es depositada por el obrero en el respectivo camión. Dado que fue escogida la caja roja y el camino A para realizar la simulación se observa que inicialmente dicha caja, que se encontraba al nivel del suelo, es elevada hasta un punto a 0.8 metros del mismo, recorre los 5.1 metros horizontales del tramo recto y es bajada para ser puesta nuevamente en el suelo. Después se observa la trayectoria inclinada que realiza la caja, indicando que en efecto está siendo arrastrada por la rampa de elevación. Finalmente, la caja es llevada desde el final del plano inclinado hasta una posición vertical a 0.8 metros de este punto.

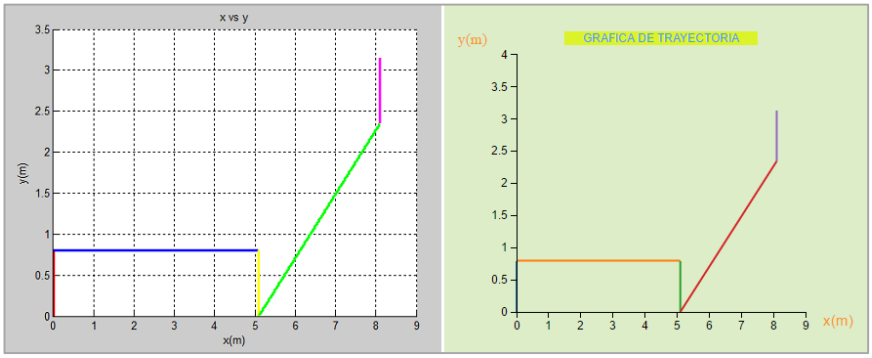

Figura 73. Gráficas de trayectoria x vs y – Trabajo y energía.

- **Gráfica del trabajo del obrero W vs x:**

Finalmente la gráfica que representa el trabajo realizado para llevar la caja por el trayecto escogido se muestra en la figura 75, partiendo de las ecuaciones (33), (34) y (35). En el punto de partida se observa que para levantar la caja el obrero efectúa un trabajo equivalente a $W = 39.2\,J$ (el desplazamiento de la caja y la fuerza aplicada tienen la misma dirección), mientras que en el movimiento por el tramo recto el trabajo realizado es considerado nulo ya que la dirección de la fuerza

aplicada para sostener dicha caja es perpendicular a la dirección de dicho movimiento. Al final de este segmento, el trabajo hecho para bajar la caja al suelo tiene un valor de $W = -39.2\,J$, debido a que las direcciones del desplazamiento y la fuerza aplicada son opuestas.

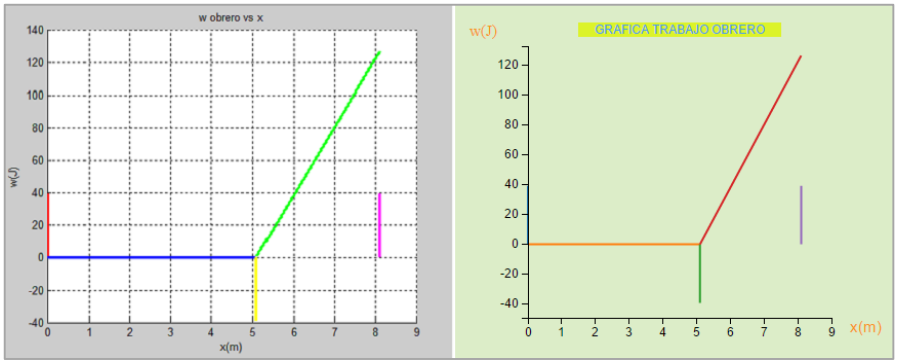

Figura 74. Gráficas de W del obrero vs t – Trabajo y energía.

En el tramo correspondiente al plano inclinado el trabajo está determinado por el cambio de energía cinética de la caja en el punto inicial y final de la rampa, el cual llega hasta un valor de $W = 126.33\,J$. El trabajo final, llevado a cabo en el proceso de levantar la caja desde el punto final de la rampa hasta el camión número 1, tiene un valor de $W = 39.2\,J$, similar al del punto inicial.

3.2.6. Práctica 6 - Colisiones:

Finalmente se presenta la interfaz principal de la práctica de colisiones (figura 75), la cual está conformada por el área de simulación donde se llevan a cabo las diferentes secuencias de animación que simbolizan el juego de billar implementado, un panel de ingreso de datos, los botones para activar los dos procesos disponibles (simulación y reiniciar) y la tabla de resultados numéricos. Debido a que la simulación debe ir de la mano con un proceso previo en donde se calcula la dirección con que el taco de billar debe apuntar a la bola objetivo, la interfaz despliega una serie de datos numéricos necesarios para la realización del cálculo respectivo. Los datos suministrados son los siguientes: Coeficiente de fricción de la mesa, duración del impulso en el impacto de la bola por parte de taco de billar, radio de las bolas, masa de las bolas, posición de la bola de impacto (bola blanca) y las posiciones de las buchacas 1 y 2.

Figura 75. Interfaz de la aplicación de movimiento en 2D.

3.3. GUÍAS DE ACTIVIDADES

El desarrollo de la herramienta de simulación incluye también la implementación de las guías de actividades, las cuales contienen los pasos necesarios para llevar a cabo cada una de las simulaciones. Estas se encuentran disponibles en el anexo B denominado “guías de actividades para la realización de las prácticas”.

3.4. EVALUACIÓN DE LA APLICACIÓN WEB

Con el fin de estimar y evaluar la percepción de los usuarios respecto a la utilización de la aplicación software se diseñó una pequeña encuesta cualitativa dirigida a un grupo de estudiantes de los cursos respectivos de física mecánica orientados dentro de los programas de ingeniería de la Universidad del Cauca. Tanto el trabajo con el simulador como la aplicación de la encuesta fueron realizados con la colaboración del ingeniero Francisco Franco Obando Díaz, quien en ese momento estaba a cargo de uno de los cursos de mecánica del programa de Automática Industrial en el transcurso del primer periodo académico de 2016. Las prácticas utilizadas dentro de las actividades propuestas por el docente como parte del desarrollo del curso fueron las siguientes:

- Práctica de vectores.
- Práctica de movimiento en dos dimensiones.
- Práctica de Leyes del movimiento.
- Práctica de trabajo y energía.

Nota: El formato de la encuesta realizada a los estudiantes del curso de mecánica está disponible en el anexo denominado “encuesta de manejo de la aplicación”.

3.4.1. Resultados de la encuesta del manejo de la aplicación.

Las encuestas de manejo de la aplicación se llevaron a cabo como parte de la actividad de evaluación de los conceptos de la asignatura realizados a través de las diferentes actividades en que se involucró la herramienta de simulación. La figura 76 muestra los resultados gráficos de la encuesta realizada:

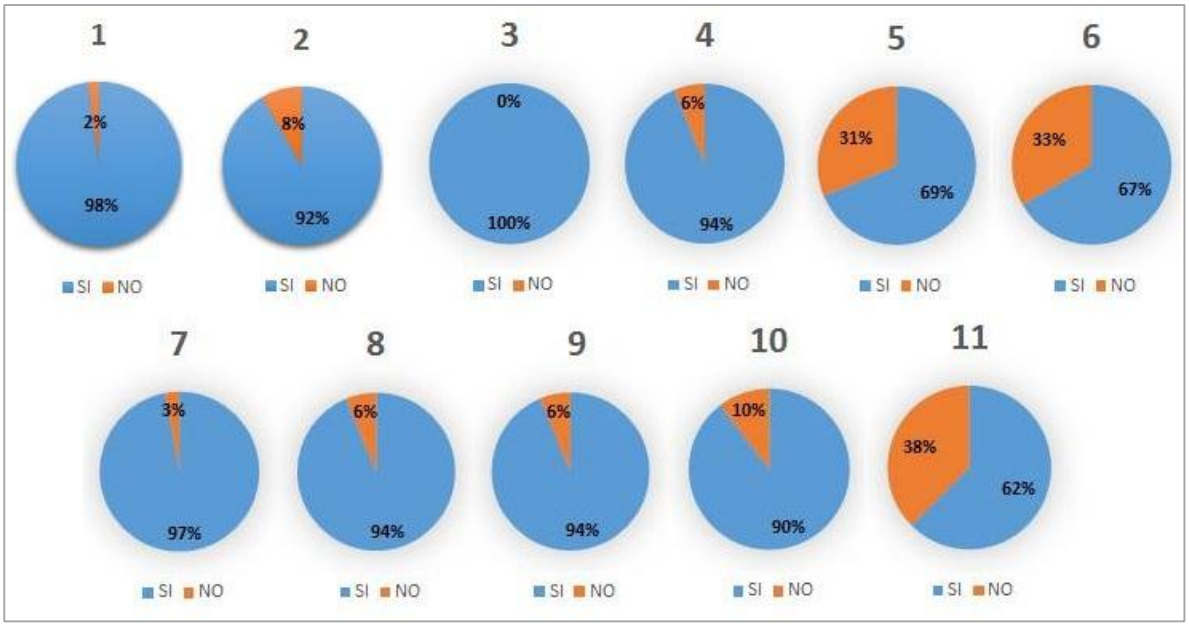

Figura 76. Resultados gráficos de la encuesta del manejo de la aplicación.

3.4.2. Análisis de los resultados obtenidos

En la figura 76 se observan los porcentajes que muestran la percepción general de los estudiantes después de haber hecho uso de la herramienta de simulación. De acuerdo a los resultados obtenidos se concluye lo siguiente:

Dentro del gráfico se observa que los estudiantes casi en su totalidad (98%) no tuvieron problema alguno en ingresar a la aplicación al seguir los pasos dados dentro de las respectivas guías de actividades, como se observa en la figura 76-1, lo que indica que los procedimientos enunciados en cada una de ellas fueron claros, teniendo un alto porcentaje a favor de 92% (ver figura 76-2). Por su parte el 100% de los encuestados confirma que la prueba de conocimiento sí está relacionada con los conceptos físicos tratados dentro de cada práctica de simulación (ver figura 76-

3) y un alto porcentaje del 94% considera que la información teórica y las ecuaciones de movimiento suministradas son de ayuda para la solución de cada una de las pruebas de conocimiento dentro de cada práctica (ver figura 76-4).

Un porcentaje del 69% afirma que el manejo de la herramienta les resultó intuitivo (ver figura 76-5), lo que se debe a que algunos procesos de simulación debían ser antecedidos por una serie de procedimientos matemáticos para determinar ciertos datos de entrada, motivo por el cual los encuestados que omitían estos pasos no podían llevar a cabo un proceso de simulación correcto. Un porcentaje del 67% afirma que la herramienta sí presentó problemas durante su ejecución (ver figura 76-6), pero de acuerdo a los comentarios dados, se trataban de inconvenientes menores que ya fueron solucionados.

En cuanto a los resultados numéricos generados por la aplicación un 97% considera que estos fueron correctamente asimilados y que concuerdan con los obtenidos matemáticamente por ellos (ver figura 76-7). De igual manera, un alto porcentaje de los encuestados equivalente al 94% confirma que las gráficas o curvas de movimiento generadas por la aplicación sí les brindaron información clara respecto a los fenómenos estudiados dentro de las respectivas prácticas de simulación (ver figura 76-8).

Por su parte un 94% de los estudiantes encuestados encuentra agradable la interacción con la herramienta de simulación (ver figura 76-9) y un grupo correspondiente al 90% considera que en efecto la aplicación cumple con su función de ser un elemento apropiado para complementar los conceptos teóricos vistos en clase (ver figura 76-10). Finalmente el 62% de los estudiantes que fueron indagados considera que se pueden hacer mejoras en cuanto al funcionamiento de la aplicación, sin embargo a través de los comentarios destacan el aporte positivo que esta puede llegar a brindar a las actividades desarrolladas dentro de los diferentes cursos de física mecánica (ver figura 76-11).

En términos generales la encuesta de manejo de la aplicación arroja unos buenos resultados, lo que confirma su impacto positivo como elemento complementario a los conceptos teóricos manejados por los estudiantes. Sin embargo, al tratarse de uno de los primeros proyectos enfocados a la realización de prácticas virtuales relacionados con esta área del conocimiento, quedan algunas cosas por mejorar e implementar a través de trabajos posteriores.

Capítulo 4

CONCLUSIONES

- La aplicación al ser implementada bajo el estándar de desarrollo web HTML5 cuenta con una serie de ventajas que potencian su utilización dentro de los cursos de física mecánica de la Universidad del Cauca, ya que, entre otras cosas, brinda a estudiantes y docentes la posibilidad de trabajar con cada una de las prácticas virtuales en cualquier momento y de forma inmediata a través de internet, sin la necesidad de instalación o actualización de elementos complementarios que puedan limitar y comprometer la realización de las diferentes actividades de experimentación.

- El lenguaje de desarrollo web HTML5 pone a disposición de los usuarios una gran cantidad de herramientas y una vasta documentación para llevar a cabo la implementación de aplicaciones diversas y con grandes características de funcionalidad de manera gratuita. En especial, cuenta con librerías y Frameworks de código abierto muy potentes que permiten el desarrollo de aplicaciones enfocadas al ámbito educativo, como el caso de Phaser, elemento con el cual fue posible llevar a cabo la construcción de cada una de los sistemas físicos a partir de animaciones en 2D con una gran calidad gráfica, propiedad que antes solo la brindaban algunos lenguajes como Adobe Flash, el cual requiere del pago de una licencia para su utilización.

- La herramienta desarrollada al ser de tipo web tiene la ventaja de ser multiplataforma y su funcionalidad no se ve afectada por el sistema operativo donde se ejecute, además de que posee características de portabilidad que le permiten correr en cualquier ordenador con conexión a internet.

- Debido a que el estándar HTML5 es relativamente nuevo algunos navegadores no soportan la totalidad de sus características, siendo el explorador Google Chrome el más compatible a la fecha, seguido por Mozilla Firefox, Opera, Safari y por último Internet Explorer. Por tal motivo se recomienda contar con la versión más actualizada del navegador de Google para trabajar con la aplicación desarrollada.

- Las herramientas de simulación puestas al servicio de la física permiten la implementación de un variado número de sistemas que son difíciles de reproducir en un laboratorio convencional y brindan la posibilidad de obtener resultados de manera rápida y con un alto grado de confiabilidad.

- Aunque en la actualidad se disponen de algunas herramientas gratuitas de simulación relacionadas con las temáticas de la física, la gran mayoría no ofrece un claro despliegue de resultados numéricos y gráficos, a pesar de contar con un buen grado de interactividad. Ante esta situación cobra una mayor importancia la herramienta implementada, ya que aparte de trabajar con sistemas físicos bien estructurados facilita la documentación necesaria para llevar a cabo un mejor proceso de simulación.

REFERENCIAS BIBLIOGRÁFICAS

[1] I. Fernández, “Las Tics En El Ámbito Educativo,” pp. 1–9, 2010.

[2] P. Camargo. (2014, Aug 13). *Las TIC como herramientas facilitadoras en la gestión pedagógica* [Online]. Available: http://www.unitecnologica.edu.co/educacionadistancia/newletter/2014/bole tin006/noti_apliaciones/005-lastic/index.html

[3] V. Duro. (2013, Jul 2). *Uso del software educativo en el proceso de enseñanza y aprendizaje* [Online]. Available: http://www.gestiopolis.com/uso-del-software-educativo-en-el-proceso-de-ensenanza-y-aprendizaje/

[4] C. Meneses and L. Artuduaga, “Software educativo para la enseñanza y aprendizaje de las matemáticas en el grado 6,” p. 92, 2014.

[5] C. Infante, “propuesta pedagógica para el uso de laboratorios virtuales como actividad como actividad complementaria en las actividades teórico-prácticas,” Revista mexicana de investigación educativa, vol. 19, pp. 917–937, 2014.

[6] D. H. Jonassen, “El diseño de entornos constructivistas de aprendizaje,” *Diseño la Instr. teorías y Model. un nuevo Paradig. la teoría la Instr.*, vol. 76, 77, pp. 225–249, 2000.

[7] A. Pontes Pedrajas, “Aplicaciones de las tecnologías de la información y de la comunicación en la educación científica. Segunda parte: aspectos metodológicos,” *Rev. Eureka sobre Enseñanza y Divulg. las Ciencias*, vol. 2, no. 3, pp. 330–343, 2005.

[8] C. Sarabando, J. P. Cravino, and A. A. Soares, “Contribution of a Computer Simulation to Students’ Learning of the Physics Concepts of Weight and Mass,” *SLACTIONS 2013 Res. Conf. virtual worlds – Learn. with simulations*, vol. 13, pp. 112–121, 2014.

[9] E. H. El Hassouny, F. Kaddari, A. Elachqar, and A. Alami, "Teaching/Learning Mechanics in High School with the Help of Dynamic Software," *5th World Conf. Educ. Sci.*, vol. 116, pp. 4617–4621, 2014.

[10] U. I. Bahytovna, B. A. Nurullaevich, B. K. Meirbekovich, and S. A. Chozhankizi, "Electronic Resources in Physics as a Means of Formation Applied Orientation of Students," *Procedia - Soc. Behav. Sci.*, vol. 116, pp. 4310–4314, 2014.

[11] T. Krobthong, "Teaching University Physics by Using Interactive Science Simulations Methods," *Procedia - Soc. Behav. Sci.*, vol. 197, no. February, pp. 1811–1817, 2015.

[12] M. Kock Pedersen, A. Svenningsen, N. B. Dohn, A. Lieberoth, and J. Sherson, "DiffGame: Game-based mathematics learning for physics," *Procedia - Soc. Behav. Sci.*, vol. 228, no. 16, pp. 316–322, 2016.

[13] G. Arias. (2016, May 8). *GNU FisicaLab: El software libre para la física* [Online]. Available: https://www.gnu.org/software/fisicalab/

[14] G. Arias. (2015, May 30). FisicaLab - News: GNU FisicaLab 0.3.5 released [Online]. Available: http://savannah.gnu.org/forum/forum.php?forum_id=8283

[15] R. Educyt, A. Hurtado 1, H. Orjuela, "acercamiento al desarrollo de software de simulación interactivo como herramienta en la enseñanza y aprendizaje de la física: proyecto step", 2012.

[16] H. J. Orjuela Ballesteros and A. Hurtado Márquez, "Perfeccionamiento de un nuevo simulador interactivo, bajo software libre gnu/linux, como desarrollo de una nueva herramienta en la enseñanza y aprendizaje de la física," *Latin-American J. Phys. Educ.*, vol. 4, no. 1, p. 30, 2010.

[17] Algoryx. (2016). *Algodoo: What is it?* [Online]. Available: http://www.algodoo.com/what-is-it/

[18] S. V Rueda, A. Cohen, T. Delladio, S. Gottifredi, and L. H. Tamargo, "Herramientas para apoyar el descubrimiento de vocaciones en Ciencias de la Computación," *XX Congr. Argentino Ciencias la Comput.*, pp. 1–10, 2014.

[19] D. Brown. (2016). Tracker: *Video Analysis and modeling tool* [Online]. Available:

http://physlets.org/tracker/

[20] Design Simulation Technologies. (2016). *Interactive physics: Physics simulation software for the classroom* [Online]. Available: http://www.design-simulation.com/IP/index.php

[21] Design Simulation Technologies. (2016). *Interactive physics: Simulations* [Online]. Available: http://www.design-simulation.com/IP/simulations.php

[22] D. Xanthopoulos. (2015, Dec 7). *Physion - Physics Simulation Software* [Online]. Available: http://physion.net/

[23] The Qt Company. (2016).Qt *is a flexible and customisable framework* [Online]. Available: https://www.qt.io/download/#Licence-anchor

[24] E. Catto. (2015, Jul 12). *Box2D:* A *2D Physics Engine for Games* [Online]. Available: http://box2d.org/

[25] D. Xanthopoulos. (2015, Dec 7). *Physion: About Physion* [Online]. Available: http://physion.net/en/about

[26] A. Ospina. (2016). Cienytec: *Laboratorio virtual / Software de física* [Online]. Available: http://www.cienytec.com/edu2_software_fisica_laboratorio_virtual.htm

[27] Go-Lab Project. (2016). *Phet Interactive Simulations* [Online]. Available: http://www.go-lab-project.eu/partner/phet-interactive-simulations

[28] Edshelf. (2016). *Phet Interactive Simulations* [Online]. Available: https://edshelf.com/tool/phet-interactive-simulations/

[29] University of Colorado. (2016). *Phet interactive simulations* [Online]. Available: https://phet.colorado.edu/es/simulations/category/physics/motion

[30] University of Colorado. (2016). *Phet interactive simulations: HTML5 sims*

[Online]. Available:
https://phet.colorado.edu/en/simulations/category

[31] T. Casellas. (2016). *Fislab.net: Laboratorio virtual de física* [Online]. Available:
http://www.fislab.net/

[32] The physics aviary. (2016). *HTML 5 Physics Lab Simulations* [Online]. Available:
http://www.thephysicsaviary.com/Physics/Programs/Labs/#mechanics

[33] The physics classrooms. (2016). *Physics interactives* [Online]. Available:
http://www.physicsclassroom.com/Physics-Interactives

[34] T. Martín and A. Serrano. (2014). *Animaciones de física* [Online]. Available:
http://acer.forestales.upm.es/basicas/udfisica/asignaturas/fisica/animaciones.html

[35] Corporación universitaria del Caribe. (2014). *Desarrollo de software educativo* [Online]. Available:
http://campusvirtual.cecar.edu.co/modulos/Tecnolog%C3%ADa%20e%20inform%C3%A1tica/7%20SEMESTRE/DESARROLLO%20DE%20SOFTWARE%20EDUCATIVO/index.html#p=6

[36] D. López. (2016, Jun 28). *Aplicaciones de escritorio vs Aplicaciones Web* [Online]. Available:
http://byspel.com/aplicaciones-de-escritorio-vs-aplicaciones-web/

[37] Clínica en la nube. (2014). *Aplicación de escritorio o nube, ventajas y desventajas* [Online]. Available:
http://www.clinicaenlanube.com/2014/06/aplicacion-escritorio-o-nube-ventajas-y-desventajas/

[38] A. Pastorini. (2014). *Introducción a RIA* [Online]. Available:
https://www.fing.edu.uy/tecnoinf/mvd/cursos/ria/material/teorico/ria-01-introduccion.pdf

[39] I. Corporation, “El Desarrollo De Aplicaciones Moviles Nativas,” p. 10, 2012.

[40] OK hosting. (2016). *Ventajas y desventajas de las aplicaciones móviles nativas* [Online]. Available:

http://okhosting.com/blog/ventajas-desventajas-de-las-aplicaciones-moviles-nativas/

[41] Zenba Pty Ltd. (2015). Tipos de aplicaciones móviles y sus características [Online]. Available: https://deideaaapp.org/tipos-de-aplicaciones-moviles-y-sus-caracteristicas/

[42] J. Gauchat. (2012, Jan 20). *El gran libro de HTML5, CSS3 y JavaScript* [Online]. Available: https://adegiusti.files.wordpress.com/2013/09/el-gran-libro-de-html5-css3-y-javascript.pdf

[43] C. Avila, “Desarrollo innovador de interfaces web utilizando tecnología HTML5 en el diseño de una aplicación web,”Tesis de maestría, Dep. computación, Universidad del Azuay, Cuenca, Ecuador, 2012.

[44] T. Abraham, R. Rodríguez, U. De Las, P. De Gran, and D. H. Romano, “Desarrollo del juego Stacker para HTML5 / Canvas,” 2014.

[45] M. Gutrina, “Aprender a programar con Cpp,” p. 235, 2010.

[46] M. León. (2012). *Unidad 1 - flash cs6* [Online]. Avaiable: https://clasesmarthaleon.files.wordpress.com/2013/05/unidad-1-flash-cs6.pdf

[47] M. Domínguez, M. A. Prada, A. Morán, S. Alonso, and P. Barrientos, “Improving user interaction in remote laboratories through HTML5/AJAX,” *IFAC Proc. Vol.*, vol. 9, no. PART 1, pp. 282–287, 2012.

[48] C. Román. (2015). *Tipos de programas en Java* [Online]. Available: http://profesores.fi-b.unam.mx/carlos/java/java_basico1_4.html

[49] M. Murguía. (2007, Aug 12). *¿Qué es un Applet?* [Online]. Available: http://campus.iztacala.unam.mx/mmrg/mono54/archivos_archivos/arch_0 0 5_applets.pdf

[50] C. Arguedas and A. Bejarano, “Uso de applets de java en el curso en el curso en línea de física II, valoración del estudiantado para su aplicación en secundaria,” Revista pedagógica Atenas, vol. no. 2, pp. 109-122, July, 2015.

[51] M. Ramos. “Simulación mediante applets de java: el comportamiento del péndulo caótico,” proyecto de fin de carrera, Dep. de física, Universidad Rey Juan Carlos, Madrid, España, 2010.

[52] Unam. (2014). *Análisis de herramientas para la implementación del sistema* [Online]. Available: http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.10 0/216/A8.pdf?sequence=8

[53] M. del P. Salas-Zárate, G. Alor-Hernández, and A. Rodríguez-González, “Developing Lift-based Web Applications Using Best Practices,” *Procedia Technol.*, vol. 3, pp. 214–223, 2012.

[54] M. del P. Salas-Zárate, G. Alor-Hernández, and A. Rodríguez-González, “Developing Lift-based Web Applications Using Best Practices,” *Procedia Technol.*, vol. 3, pp. 214–223, 2012.

[55] L. de castro, A. García and M. Hernández, “Videojuegos para aprender a programar videojuegos,” Trabajo de fin de grado, Dep. Ingeniería de software e inteligencia artificial, Universidad complutense de Madrid, Madrid, España, 2015.

[56] A. Lago, A. L. Ferreiro, A. R. Simón, S. L. Casas, A. A. Nogueiras, and J. M. Acevedo, “Herramienta educativa para el estudio de Circuitos de Control en Modo Tensión de,” no. July, 2016.

[57] B. Van Damme and R. Van der Linde, “Por qué a los desarrolladores de juegos deberia interesarles html5,” Revista Novática, vol. 230, pp. 25-31, October-December, 2014.

[58] D. Cap, “Ciclo de vida del software,” *Int. Organ.*, pp. 2005–2005, 2005.

[59] J. D. Alemán Suárez and M. A. Mata Mendoza, “Guía de elaboración de un manual de prácticas de laboratorio, taller o campo: Asignaturas teórico prácticas,” pp. 1–28, 2006.

[60] J. Paredes. (2015, Jul 22). *Programación de juegos con HTML5 y JavaScript* [Online]. Available: http://joelwalls.com/2015/07/programacion-de-juegos-con-html5-y-javascript/?i=1

[61] J. Nicholls. (2015, Jan 28). *Creando juegos HTML5 con Phaser Framework en Mónaco, el editor de código de Visual Studio Online* [Online]. Available: http://www.nicholls.co/blog/post/Creando-Juegos-HTML5-con-Phaser-en-Monaco

[62] AbelRL. (2012). *¿Cómo agregar soporte WebGL a Google Chrome?* [Online]. Available: http://menteprincipiante.com/2010/12/como-agregar-soporte-webgl-a-google-chrome/

[63] Pivotstudio.co. (2015). *¿Qué es un Sprite, Sprite Sheet y un Atlas?* [Online]. Available: http://www.pivotstudio.co/blog/que-es-un-sprite-sprite-sheet-atlas

[64] Phaser.io. (2016). *Phaser API Documentation, Animation* [Online]. Available: http://phaser.io/docs/2.6.2/index#animation

[65] Phaser.io. (2016). *Class: Phaser.BitmapData* [Online]. Available: http://phaser.io/docs/2.6.2/Phaser.BitmapData.html

[66] Phaser.io. (2016). *Class: Phaser. Math, linearInterpolation* [Online]. Available: http://phaser.io/docs/2.6.2/Phaser.Math.html#linearInterpolation

[67] R. Davey. (2015). *Sprite Motion Paths Tutorial* [Online]. Available: http://phaser.io/tutorials/coding-tips-008

[68] N. Prado. (2015). *Animación con Canvas de HTLM5* [Online]. Available: https://devcode.la/tutoriales/animacion-con-canvas-de-html5/

[69] A. Guevara. (2015). *Introducción a Canvas de HTML5* [Online]. Available: https://devcode.la/tutoriales/introduccion-a-canvas-de-html5/

[70] Hipertextual.com. (2014, Jul 26). *5 librerías JavaScript para crear diagramas y gráficas sin mucho esfuerzo* [Online]. Available: https://hipertextual.com/archivo/2014/07/librerias-javascript-para-graficos/

[71] Emezeta.com. (2014). *Guía de sublime text, ¿el mejor editor de código*? [Online]. Available: http://www.emezeta.com/articulos/guia-sublime-text

[72] Medibang Paint. (2016). *Medibang Paint pro, la versión en computador de la serie Medibang Paint* [Online]. Available: https://medibangpaint.com/es/pc/

[73] M. Lacono. (2016). *Programación de videojuegos: Desarrolla tu propio proyecto en JavaScript y HTML5* [Online]. Available: https://books.google.com.co/books?id=KC2mDAAAQBAJ&pg=PT282&lpg=PT282&dq=phaser+canvas+y+phaser+auto&source=bl&ots=ZqXOtEytvV&sig=BUKf6zCdkw6TlTDarAdpGGkFpec&hl=es&sa=X&ved=0ahUKEwjnrOG8rMvOAhXHSSYKHXw2BSYQ6AEIUzAH#v=onepage&q=phaser%20canvas%20y%20phaser%20auto&f=false

[74] A. Ourrad and R. Davey. (2013). *Making your first Phaser game* [Online]. Available: http://phaser.io/tutorials/making-your-first-phaser-game/part4

[75] WowSlider. (2016). *WowSlider: Generador Drag-n-Drop para Windows y Mac* [Online]. Available: http://wowslider.com/es/