

**ESCUELA MILITAR DE INGENIERÍA
MCAL. ANTONIO JOSÉ DE SUCRE
“BOLIVIA”**

TRABAJO DE GRADO



**DESARROLLO DE UN SIMULADOR PARA EL
ENTRENAMIENTO DE TIRO CON FUSILES DE ASALTO FN
FAL, GALIL AR Y AK-47, BAJO ENTORNOS DINÁMICOS Y
BLANCOS MÓVILES USANDO REALIDAD VIRTUAL.**

**CASO DE ESTUDIO: BATALLÓN POLICÍA MILITAR III
“ESTEBAN ARCE”**

CARLOS ANDRES CRISNER VELARDE

COCHABAMBA, 2020

**ESCUELA MILITAR DE INGENIERÍA
MCAL. ANTONIO JOSÉ DE SUCRE
“BOLIVIA”**

TRABAJO DE GRADO

**DESARROLLO DE UN SIMULADOR PARA EL
ENTRENAMIENTO DE TIRO CON FUSILES DE ASALTO FN
FAL, GALIL AR Y AK-47, BAJO ENTORNOS DINÁMICOS Y
BLANCOS MÓVILES USANDO REALIDAD VIRTUAL.**

CARLOS ANDRES CRISNER VELARDE

**Modalidad: Trabajo de Grado
presentado como requisito
para optar al título de
Licenciatura en Ingeniería de
Sistemas**

TUTOR: ING. CLAUDIA UREÑA HINOJOSA

COCHABAMBA, 2020

DEDICATORIA

Este trabajo está dedicado a:

A mi familia por brindarme su apoyo incondicional durante mi etapa de estudiante universitario.

Finalmente, a todas aquellas personas que me brindaron su apoyo moral e intelectual para lograr culminar con éxito esta etapa de mi vida.

AGRADECIMIENTOS

A: Mis padres, los cuales fueron el pilar fundamental durante mi trayectoria de estudiante, apoyándome de manera incondicional.

A: Cnl. DAEN. Tomas Crisner Prado, gracias por apoyarme en todo momento de manera incondicional y ayudarme a concluir esta etapa de mi vida.

A: Sra. Gladys Velarde de Crisner, gracias por el apoyo incondicional que me brindaste a lo largo de mi etapa como estudiante.

A: My. DIM. Camilo Rios Aguilar, gracias por su tiempo y su apoyo como jefe de la carrera de ingeniería de sistemas.

A: Ing. Cesar Jhoel Tamares Terrazas, gracias por apoyarme con los medios de hardware, los cuales fueron vitales para la realización de mi trabajo de grado.

A: Ing. Claudia Ureña Hinojosa, gracias por guiarme en esta fase final de mi vida estudiantil.

A: Ing. Sheyla Salvatierra Rojas, gracias por tu tiempo y ayuda incondicional que me brindaste en el transcurso de mi vida universitaria.

A: A todas las personas que estuvieron presentes en mi trayecto de estudiante, y en especial a Bruno que estuvo presente todo el tiempo, durante la realización de este trabajo de grado.

A: Escuela Militar de Ingeniería, en la cual se me brindaron muchas enseñanzas, oportunidades y a su vez lindas experiencias, también preparándome para mi futuro como ingeniero de sistemas, en la ciudad tecnológica de Bolivia.

RESUMEN EJECUTIVO



TRABAJO DE GRADO
INGENIERIA DE SISTEMAS
DESARROLLO DE UN SIMULADOR PARA EL ENTRENAMIENTO DE TIRO CON
FUSILES DE ASALTO FN FAL, GALIL AR Y AK-47, BAJO ENTORNOS
DINÁMICOS Y BLANCOS MÓVILES USANDO REALIDAD VIRTUAL.
AUTOR: CARLOS ANDRES CRISNER VELARDE
TUTOR: LIC. CLAUDIA UREÑA HINOJOSA

RESUMEN EJECUTIVO

Para la siguiente tesis de grado, bajo la modalidad de tesis, se desarrolló la herramienta DESARROLLO DE UN SIMULADOR PARA EL ENTRENAMIENTO DE TIRO CON FUSILES DE ASALTO FN FAL, GALIL AR Y AK-47, BAJO ENTORNOS DINÁMICOS Y BLANCOS MÓVILES USANDO REALIDAD VIRTUAL.

El presente trabajo, está dividido en 5 capítulos, los cuales se describen a continuación:

Este trabajo de grado, busco demostrar la reducción de costos de entrenamiento de tiro con fusiles, utilizando el simulador. Así poder llegar al objetivo y desglosar en 5 capítulos con sus respectivos procesos.

Como inicio, se llevó a cabo el primer capítulo de generalidades, en el cual se presentó la formulación del problema y por qué se está realizando este simulador, y como podría ser resuelto mediante el objetivo general que se ha planteado y los objetivos específicos, los cuales fueron realizados durante todo el desarrollo de la tesis de grado.

Como capítulo II se trabajó el marco teórico; en este capítulo, se tomó en cuenta todos los objetivos específicos planteados y que la teoría fue necesaria para la realización de cada objetivo específico, siendo un respaldo para proceder a la parte práctica.

En el capítulo III denominado el marco práctico, se inició con la implementación de los métodos procedurales para la generación del entorno dinámico en base a imágenes satelitales; también se efectuó el modelado 3d de los objetos utilizables y de entorno del simulador, para finalmente pasar a la programación e integración de todos los módulos del simulador. De igual forma, se desarrolló la demostración de hipótesis, dando lugar a que la solución planteada mejore el problema identificado.

Para el capítulo IV, se hizo el análisis de la viabilidad técnica y económica, donde se presentó los requerimientos de hardware y sus respectivos costos; también se calculó los costos de personal y con esta información se aplicó costo-beneficio que llegó a resultar viable para su implementación

Finalmente, en el capítulo V, se realizó las respectivas recomendaciones a esta tesis de grado y se desarrolló las conclusiones obtenidas durante todo el transcurso de la misma.

Las conclusiones muestran el cumplimiento de los objetivos específicos y así mismo las recomendaciones pertinentes a la implementación del presente trabajo.

Palabras claves: RV (realidad virtual) Entornos dinámicos Procedural

ABSTRACT



DEGREE WORK
SYSTEMS ENGINEER

**DEVELOPMENT OF A SIMULATOR FOR SHOOTING TRAINING WITH FN FAL,
GALIL AR AND AK-47 ASSAULT RIFLES, UNDER DYNAMIC ENVIRONMENTS
AND MOBILE TARGETS USING VIRTUAL REALITY.**

AUTHOR: CARLOS ANDRES CRISNER VELARDE

TUTOR: LIC. CLAUDIA UREÑA HINOJOSA

ABSTRACT

For the next degree thesis, under the thesis modality, the DEVELOPMENT OF A SIMULATOR FOR SHOOTING TRAINING WITH FN FAL, GALIL AR AND AK-47 ASSAULT RIFLES, UNDER DYNAMIC ENVIRONMENTS AND MOBILE TARGETS USING VIRTUAL REALITY was developed.

This work is divided into 5 chapters, which are described below:

This degree work, I seek to demonstrate the cost reduction of rifle shooting training, using the simulator. So, to be able to reach the objective and break it down into 5 chapters with their respective processes.

As a beginning, the first chapter of generalities was carried out, in which the formulation of the problem was presented and why this simulator is being carried out, and how it could be solved by means of the general objective that has been set and the specific objectives, the which were carried out throughout the development of the degree thesis.

As chapter II the theoretical framework was worked; In this chapter, all the specific objectives proposed were taken into account and that the theory was necessary for the realization of each specific objective, being a support to proceed to the practical part.

In chapter III called the practical framework, it began with the implementation of procedural methods for the generation of the dynamic environment based on satellite images; The 3d modeling of the usable objects and the simulator environment was also carried out, to finally proceed to the programming and integration of all the simulator modules. In the same way, the hypothesis demonstration was developed, resulting in the proposed solution improving the identified problem.

For chapter IV, the analysis of the technical and economic feasibility was made, where the hardware requirements and their respective costs were presented; Personnel costs were also calculated and with this information cost-benefit was applied, which became feasible for its implementation.

Finally, in chapter V, the respective recommendations to this degree thesis were made and the conclusions obtained throughout the course were developed.

The conclusions show the fulfillment of the specific objectives and also the pertinent recommendations for the implementation of this work.

Keywords: VR (virtual reality)

Dynamic environments

Procedural

ÍNDICE



ÍNDICE DE CONTENIDO

CONTENIDO	PÁGINAS
CAPITULO 1: GENERALIDADES.	1
1.1. INTRODUCCIÓN.....	1
1.2. ANTECEDENTES.	2
1.3. PLANTEAMIENTO DEL PROBLEMA.....	4
1.3.1. Identificación del problema.	4
1.3.2. Formulación del problema.	5
1.4. Objetivos y acciones.....	6
1.4.1. Objetivo general.	6
1.4.2. Objetivos específicos y acciones de la investigación.	6
1.5. JUSTIFICACIÓN	10
1.5.1. Justificación técnica.....	10
1.5.2. Justificación económica.....	10
1.5.3. Justificación operativa.	11
1.6. DELIMITACIÓN DE LA INVESTIGACIÓN.	11
1.6.1. Delimitación temática.....	11
1.6.2. Delimitación espacial.	11
1.6.3. Delimitación temporal.	11
1.7. HIPÓTESIS.	12
1.7.1. Identificación de las variables.	12
1.7.2. Definición conceptual de las variables.	12
1.7.3. Operativización de las variables.	13
1.8. MATRIZ DE CONSISTENCIA.....	13
CAPITULO 2: MARCO TEÓRICO	14
2.1. ESQUEMA DEL MARCO TEÓRICO	14
2.2. REALIDAD VIRTUAL	18

2.2.1.	Hardware.....	19
2.2.2.	Software.	30
2.2.3.	Interacción.....	35
2.2.4.	Evaluación de sistemas de realidad virtual y experiencias.....	44
2.2.5.	Generación de terreno procesal.	46
2.3.	UNITY.	50
2.3.1.	Importado.	50
2.3.2.	Manejo de materiales y texturas.	55
2.3.3.	Entradas.	56
2.3.4.	Físicas.....	63
2.3.5.	Scripting.	70
2.3.6.	Audio.	72
2.4.	MODELADO 3D CON BLENDER.....	74
2.4.1.	Bloqueando.	74
2.4.2.	Horneado de texturas.	78
2.4.3.	Materiales.....	84
2.5.	FÍSICA.....	95
2.5.1.	Cinemática.	95
2.5.2.	Ley de la inercia.	97
2.5.3.	Segunda ley de Newton.....	98
2.5.4.	Tercera ley de Newton.....	98
2.6.	INGENIERÍA DE SOFTWARE.....	100
2.6.1.	Metodologías Ágiles.	100
2.6.2.	Feature Driven Development (FDD).	100
2.6.3.	Arquitecturas de software.	106
2.6.4.	Quality assurance.	113
2.7.	SISTEMAS DISTRIBUIDOS.	118
2.7.1.	SISD.....	119

2.7.2.	MISD	121
2.7.3.	SIMD	121
2.7.4.	MIMD.....	122
2.7.5.	Memoria compartida	123
2.8.	BASES DE DATOS	124
2.8.1.	Creación de tablas en SQL.....	124
2.8.2.	Especificación de los tipos de datos en una columna	124
CAPITULO 3: MARCO PRÁCTICO		129
3.1.	ANÁLISIS DEL PROCESO ACTUAL DE ENTRENAMIENTO CON FUSIL Y SU ENTORNO DE DESEMPEÑO.	129
3.1.1.	Recopilación de información del proceso entrenamiento de tiro con fusil en el ejército boliviano.	129
3.1.2.	Recopilación de información sobre los fusiles utilizados en los entrenamientos de tiro con fusil.	129
3.1.3.	Recolección de información sobre los diferentes entornos geográficos donde se realizan los entrenamientos de tiro con fusil.	132
3.2.	APLICACIÓN DEL MÉTODO FDD (FEATURE-DRIVEN-DEVELOPMENT) PARA DESARROLLAR EL SIMULADOR SEGÚN LA ARQUITECTURA HLA (HIGH LEVEL ARCHITECTURE).....	133
3.2.1.	Realización de un diagrama de objetos sobre el funcionamiento del simulador para el entrenamiento de tiro con fusiles.	134
3.2.2.	Construcción de una lista de características del simulador para el entrenamiento de tiro con fusiles.	135
3.2.3.	Agrupación de las características del simulador para el entrenamiento de tiro con fusiles en áreas subjetivas.	135
3.2.4.	Planificación en base de cada característica del simulador para el entrenamiento de tiro con fusiles.	136
3.2.5.	Construcción en base de cada característica del simulador para el entrenamiento de tiro con fusiles.	140
3.3.	IMPLEMENTACIÓN DE LOS ENTORNOS DINÁMICOS DE REALIDAD VIRTUAL EN TIEMPO REAL A TRAVÉS DE LOS MÉTODOS PROCEDURALES.	140

3.3.1.	Obtención de las curvaturas del entorno mediante el GDEM (Global Digital Elevation Map).	141
3.3.2.	Codificación del método procedural con las imágenes obtenidas mediante las coordenadas para la generación de terrenos en tiempo real.....	150
3.3.3.	Verificación del funcionamiento del método procedural y su funcionalidad a la hora de generar el terreno en tiempo real.....	153
3.4.	ESTABLECIENDO LAS CARACTERÍSTICAS DEL DISPARO EN LOS CLIMAS DIVERSOS DE ACUERDO A LOS TERRENOS DINÁMICOS.	158
3.4.1.	Recopilación de información de las variables climatológicas.	158
3.4.2.	Codificación de los atributos del clima relacionado al entorno.	161
3.4.3.	Codificación de la física relacionada a la trayectoria de disparo.	163
3.5.	IMPLEMENTACIÓN DE UNA TABLA ESTADISTICA DE TIRO PARA OBSERVAR LA EVOLUCIÓN DEL USUARIO.	165
3.5.1.	Diseño de la base de datos (SQL) para el almacenamiento de datos estadísticos por usuario.	165
3.5.2.	Creación de la base de datos (SQL).....	166
3.5.3.	Estableciendo las tablas y relaciones de la base de datos.	166
3.5.4.	Definición de los parámetros estadísticos a ser tomados en cuenta para la tabla.	168
3.5.5.	Codificación de la tabla estadística en base al usuario y los parámetros dados anteriormente.	168
3.5.6.	Generación de los reportes por usuario.....	170
3.6.	MODELACIÓN 3D DE LOS FUSILES PROPUESTOS (FN FAL, GALIL AR Y AK-47).	171
3.6.1.	Diseño de los fusiles FN FAL, GALIL AR y AK-47.	171
3.6.2.	Aplicación de las texturas respectivas en los modelos 3D de los fusiles.	175
3.6.3.	Codificación del funcionamiento de las animaciones de los modelos 3D.....	178
3.7.	VALIDACIÓN DEL SISTEMA MEDIANTE UN PLAN DE QUALITY ASSURANCE (QA).....	179

3.7.1.	Realización de las pruebas de testeo unitario al simulador.....	179
3.7.2.	Realización de las pruebas de estrés al simulador.	181
3.7.3.	Realización del test de integración a los módulos probados en el testeo unitario.	183
3.8.	DEMOSTRACIÓN DE HIPÓTESIS.	186
CAPITULO 4: ANÁLISIS DE VIABILIDAD		187
4.1.	VIABILIDAD TÉCNICA.....	187
4.2.	VIABILIDAD ECONÓMICA.....	189
4.2.1.	Estimación de costos de la empresa.	189
4.2.2.	Estimación de esfuerzo	190
4.2.3.	Estimación del costo	190
4.2.4.	Beneficio costo.....	190
CAPITULO 5: CONCLUSIONES Y RECOMENDACIONES		193
5.1.	CONCLUSIONES	193
5.2.	RECOMENDACIONES	194
BIBLIOGRAFÍA.....		197
ANEXOS		

ÍNDICE DE FIGURAS

CONTENIDO	PÁGINAS
Figura 1: Una perspectiva en tercera persona de un sistema de realidad virtual	20
Figura 2: El cerebro (y las partes del cuerpo) controlan la configuración de los órganos sensoriales.....	21
Figura 3: Se representan las "vistas" apropiadas de este mundo virtual a la pantalla	21
Figura 4: El cerebro es "engañado" para creer que el mundo virtual.....	22
Figura 5: (a) Sistema cave. (b) Auricular VR.	24
Figura 6: (a) El Sistema Touch X by 3D systems. (b) Algunos de los controladores de juego que ocasionalmente vibran.....	25
Figura 7: Las unidades de medición INERCIAL (IMU).....	26
Figura 8: (a) El sensor kinect de microsoft. (b) La profundidad se determina	27
Figura 9: (a) Google Cardboard. (b) Samsung Gear VR	29
Figura 10: Desmontaje de los auriculares oculus rift dk2	29
Figura 11: El generador virtual del MUNDO (VWG)	31
Figura 12: La zona coincidente	32
Figura 13: Espectro de escenarios de locomoción comunes.....	36
Figura 14: Locomoción a lo largo de un terreno horizontal	37
Figura 15: (a) Una cinta de correr omnidireccional utilizada en un sistema CAVE (cueva) por el ejército de los EE. UU. para entrenamiento. (b) Un sistema para montar bicicleta en casa conectado a un auricular VR.....	38
Figura 16: Teletransportación	40
Figura 17: Definir una cuenca de atracción	42
Figura 18: (a) Mandos oculus rift. (b) Utilización del Oculus Rift	43
Figura 19: Ejemplos de diagramas de Voronoi.....	48
Figura 20: El mapa de altura combinado antes de la perturbación (izquierda) y después (derecha).....	50
Figura 21: La estructura básica de archivos de un proyecto de Unity.....	51
Figura 22: La ventana del proyecto muestra los activos que Unity importó en su proyecto.....	51

Figura 23: La relación entre la carpeta de activos en su proyecto Unity en su computadora y la ventana proyecto dentro de Unity	51
Figura 24: Configuración de importación para ese activo en el inspector	52
Figura 25: Un activo de audio seleccionado en la ventana del proyecto	53
Figura 26: Articulaciones.....	69
Figura 27: Sorround	72
Figura 28: Modo edición.....	75
Figura 29: Modificadores.....	76
Figura 30: Personalización de la vista 3D	78
Figura 31: Configuración del modo de horneado.....	80
Figura 32: Oclusión ambiental en objeto 3d	81
Figura 33: Tipos de mapa de textura.....	83
Figura 34: Propiedades del material.....	85
Figura 35: Configuración del material.....	87
Figura 36: Sombreado	90
Figura 37: Editor de nodos	91
Figura 38: Glass BSDF	92
Figura 39: Mix shader	93
Figura 40: Texturas con ciclos	94
Figura 41: Combinación de materiales	94
Figura 42: Descomposición de las fuerzas en un plano inclinado.	100
Figura 43: Los cinco procesos de FDD con sus salidas	101
Figura 44: Diagrama de flujo del desarrollo de un modelo general.....	102
Figura 45: Diagrama de flujo de la construcción de una lista de características.	103
Figura 46: Diagrama de flujo para planificar por característica.....	104
Figura 47: Diagrama de flujo para diseñar por característica.	105
Figura 48: Diagrama de flujo para construir por característica	106
Figura 49: Proceso de desarrollo (HLA)	112
Figura 50: Las reglas de HLA.....	113
Figura 51: Clasificación Flynn	119

Figura 52: El esquema de arquitectura SISD	120
Figura 53: Componentes de la CPU en la fase FETCH-DECODE-EXECUTE	120
Figura 54: El esquema de arquitectura MISD.....	121
Figura 55: El esquema de arquitectura MIMD	123
Figura 56: El esquema de arquitectura de memoria compartida	124
Figura 57: Polígono de tiro utilizado por la fábrica boliviana de municiones FBM.	132
Figura 58: Polígonos de tiro en Cotapachi	133
Figura 59: Diagrama de objetos	134
Figura 60: Burn-up chart del simulador de tiro con fusil.....	139
Figura 61: Método Parking Lot FDD	140
Figura 62: Mapa de Cochabamba con divisiones por sectores.	141
Figura 63: Sector 1.....	142
Figura 64: Sector 2.....	143
Figura 65: Sector 3.....	144
Figura 66: Sector 4.....	145
Figura 67: Sector 5.....	146
Figura 68: Sector 6.....	147
Figura 69: Sector 7.....	148
Figura 70: Sector 8.....	149
Figura 71: Métodos de Unity C#.....	150
Figura 72: Habilitación de permisos de escritura y lectura.....	151
Figura 73: Nearest Neighbor Python	152
Figura 74: Nearest Neighbor C#.....	153
Figura 75: Sector 1 renderizado	154
Figura 76: Sector 2 renderizado	154
Figura 77: Sector 3 renderizado	155
Figura 78: Sector 4 renderizado	155
Figura 79: Sector 5 renderizado	156
Figura 80: Sector 6 renderizado	156
Figura 81: Sector 7 renderizado	157

Figura 82: Sector 8 renderizado	157
Figura 83: Altitud obtenida con el coprocesador M11	158
Figura 84: Temperatura de Cochabamba.....	159
Figura 85: Velocidad del viento en Cochabamba	160
Figura 86: Dirección del viento en Cochabamba	161
Figura 87: Trayectoria del proyectil influida por la fuerza del viento	162
Figura 88: Trayectoria del proyectil influida por la fuerza del viento	162
Figura 89: Movimiento parabólico	163
Figura 90: Propiedades del fusil de asalto.....	164
Figura 91: Propiedades del proyectil.	164
Figura 92: Diseño de la DB	165
Figura 93: Creación de la base de datos con SQLITE.....	166
Figura 94: Archivo de la base de datos.	166
Figura 95: Creación de tabla “Usuarios”	167
Figura 96: Creación de tabla “Puntuación”	167
Figura 97: Tabla de puntuaciones antes de la ejecución del simulador	169
Figura 98: Tabla de puntuación en Unity	169
Figura 99: Reporte generado por usuario.....	171
Figura 100: Cuerpo básico del fusil	172
Figura 101: Cuerpo del fusil de asalto AK-47.	172
Figura 102: Cargador del fusil de asalto ak-47	173
Figura 103: Cerrojo del fusil de asalto ak-47	174
Figura 104: Modelado 3d del fusil FN FAL	174
Figura 105: Modelado del fusil GALIL AR	175
Figura 106: Aplicado de textura en el fusil AK-47	175
Figura 107: Textura del fusil AK-47	176
Figura 108:Textura aplicada al fusil FN FAL	176
Figura 109:Textura del fusil FN FAL.....	177
Figura 110: Textura aplicada al fusil GALIL AR.....	177
Figura 111: Textura del fusil GALIL AR	178
Figura 112: Animación de retroceso de las armas.	179

Figura 113: Pruebas unitarias	180
Figura 114: Frecuencia del procesador máquina de prueba.....	182
Figura 115: Estadísticas obtenidas por parte del hardware en la ejecución del simulador.	182
Figura 116: Folder de Scripts	183
Figura 117: Folder scripts.....	184
Figura 118: Folder tests	184
Figura 119: Prueba al módulo de conexión de la BD	185
Figura 120: Estado del dispositivo y sus sensores.	185

ÍNDICE DE TABLAS

CONTENIDO	PÁGINAS
Tabla 1: Objetivos específicos y acciones.....	7
Tabla 2: Operativización de variables.	13
Tabla 3: Matriz de consistencia.	13
Tabla 4: Asignaciones de entrada XR	58
Tabla 5: Características del dispositivo de entrada	59
Tabla 6: Dispositivos de entrada por Rol.....	60
Tabla 7: Acceso a dispositivos de entrada por nodo XR	60
Tabla 8: Se produce la detección de colisión y se envían mensajes tras la colisión.	67
Tabla 9: Los mensajes de activación se envían en caso de colisión.	68
Tabla 10: Tabla de articulaciones.....	70
Tabla 11: Tiempos de ejecución.....	181
Tabla 12: Estadísticas tabuladas: simulador, costo operación	186
Tabla 13: Requerimientos mínimos para el desarrollo del sistema.....	187
Tabla 14: Requisitos recomendados para el uso del sistema.....	188
Tabla 15: Costo del equipo para el uso del sistema.	189
Tabla 16: Costo del entrenamiento de un batallón con munición 7.62mm.....	190
Tabla 17: Costo del entrenamiento de un batallón con munición 5.56mm.....	191
Tabla 18: Costo con el simulador de tiro con fusiles.....	191

ÍNDICE DE CUADROS

CONTENIDO	PÁGINAS
Cuadro 1: Características técnicas del fusil de asalto AK-47.....	130
Cuadro 2: Características técnicas del fusil de asalto FN FAL	130
Cuadro 3: Características técnicas del fusil de asalto GALIL AR	131
Cuadro 4: Lista de características del simulador para el entrenamiento de tiro con fusiles	135
Cuadro 5: Lista de características del simulador para el entrenamiento de tiro con fusiles	135
Cuadro 6: Características del simulador.....	137
Cuadro 7: Esfuerzo del simulador de tiro.	138

ÍNDICE DE ANEXOS

ANEXO “A”: CARTA DE ACEPTACIÓN

ANEXO “B”: CARTA DE CONFORMIDAD

ANEXO “C”: ENTREVISTA SOBRE EL PROCESO DE ENTRENAMIENTO DE TIRO

ANEXO “D”: COSTO DE UNA WORKSTATION

ANEXO “E”: CALCULO COMPLEMENTARIO HIPOTESIS

ANEXO “F”: SCRIPT PARA LOS PARAMETROS DEL ARMA

ANEXO “G”: ARCHIVO SQLITECONTROLLER

ANEXO “H”: SCRIPT PARA CREACIÓN DE TABLA DINÁMICA

ANEXO “I”: SCRIPT PARA GENERAR REPORTES

ANEXO “J”: MODELOS 3D EXTRAS

ANEXO “K”: DIAGRAMA DEL CIRCUITO E IMPLEMENTACIÓN

ANEXO “L”: CÓDIGO ARDUINO

CAPÍTULO I: GENERALIDADES



CAPITULO 1: GENERALIDADES.

1.1. INTRODUCCIÓN.

La realidad virtual (VR) es una tecnología con la cual podemos interactuar en entornos virtuales, brindándonos una inmersión dentro de estos ambientes artificiales. Al estimular artificialmente nuestros sentidos, nuestros cuerpos se ven engañados para aceptar otra versión de la realidad. La Realidad Virtual (VR) es como un sueño despierto que podría tener lugar en un mundo mágico de dibujos animados, o podría transportarnos a otra parte de la Tierra o el universo. (Steven M. LaValle, 2019, pp: 1).

Para poder aprovechar de mejor manera la realidad virtual, es necesaria la utilización de hardware que nos permita recrear las acciones y sensaciones dentro de este entorno artificial. Al considerar un sistema de realidad virtual, es tentador enfocarse solo en las partes de ingeniería tradicionales: hardware y software. Sin embargo, es igualmente importante, si no más importante, comprender y explotar las características de la fisiología y la percepción humanas, diseñando nosotros mismos estos campos se pueden considerar como ingeniería inversa. Todas estas partes se ajustan perfectamente para lograr que forme ingeniería de percepción (Steven M. LaValle, 2019, pp: 2).

La realidad virtual se clasifica en tres tipos principales: (a) Sistemas de realidad virtual no inmersiva, (b) Sistemas de realidad virtual semi-inmersiva y (c) Sistemas de realidad virtual inmersiva (totalmente inmersiva). Otras formas de clasificación son los niveles de VR y los métodos de VR. Los niveles de VR se relacionan con los esfuerzos empleados para desarrollar la tecnología VR. Bajo esta clasificación tenemos nivel de entrada, nivel básico, nivel avanzado, sistemas inmersivos y

sistemas de gran tamaño. Los métodos de clasificación de VR se ocupan de los métodos empleados en el desarrollo del sistema de VR. Bajo esta clase tenemos sistemas basados en simulación, sistemas basados en proyectores, sistemas basados en imágenes de avatar y sistemas basados en escritorio. (Moses Okechukwu Onyesolu, 2006, pp: 57)

Teniendo en cuenta lo mencionado anteriormente, podemos relacionar a la simulación, el cual tiene un parecido en cuanto a lo que se quiere lograr “La simulación es la experimentación con un modelo. El comportamiento del modelo imita algunos aspectos sobresalientes del comportamiento del sistema en estudio y el usuario experimenta con el modelo para inferir este comportamiento.” (M. D. Rossetti, 2009, pp: 12).

Aplicando realidad virtual y simulación, se puede generar una mejor inmersión dentro de nuestro entorno virtual.

En base a estos recursos tecnológicos se plantea generar un simulador de tiro con fusiles de asalto para el personal del “BATALLÓN DE POLICÍA MILITAR III ESTEBAN ARCE”.

1.2. ANTECEDENTES.

Durante la presidencia del Gral. Ismael Montes el año 1908 se implantó el servicio militar obligatorio, declarándose con dicha ley, la creación de un órgano de policía militar que sea el encargado de hacer cumplir el servicio militar obligatorio. El año 1932 surgió la necesidad de contar con unidades de policía militar que contribuyan a control del personal que se movilizaba al Teatro de operaciones, durante la ejecución de los movimientos previos. Empleando fracciones de tropa instruida para custodia y escolta de camiones que se dirigían al frente, así como para los trenes que realizan transporte de personal desde diferentes guarniciones, de esta manera se organizó la instrucción en Cotapachi (Quillacollo, Cochabamba) siendo este el primer curso formal de Policía Militar. (Ejército de Bolivia, 2018)

En esta unidad militar, durante el primer periodo de instrucción se realiza la instrucción básica individual, culminando esta con las lecciones de tiro con fusil,

mientras que, durante el segundo periodo de instrucción, se realiza la especialización, en este caso la especialidad de policía militar.

Para los entrenamientos de tiro con fusiles de asalto en los polígonos de tiro los cuales se dividen en 9 carriles, con una distancia desde el servicio STAN hasta el servicio de blancos de 300, 200 y 100 metros, durante este entrenamiento se requiere de al menos 15 personas, indistinto de los soldados que realizan sus prácticas de tiro, mencionando a continuación el número requerido del personal que desempeñan las siguientes funciones: director de tiro, furriel escribiente, 2 radio operadores, amunicionador, jefe de marcadores y servicio de blancos donde este último necesitará al menos de una escuadra de soldados para ayudar a verificar la puntuación y calificaciones de los soldados practicantes. Al momento de realizar el entrenamiento es necesario que un instructor se posicione junto a cada soldado que será evaluado para dar instrucciones de posicionamiento y voces de mando. (Diana A. Muriel Soto, 2017)

Para la realización de prácticas de tiro utilizando el fusil FN FAL cada soldado es dotado con un cargador con 20 cartuchos calibre 7.62x51mm. En caso de utilizar el fusil Galil AR el soldado es dotado con 30 cartucho calibre 5.56x45mm.

La munición mencionada anteriormente es denominada como “Munición de infantería” el cual es considerado calibre de guerra, siendo un estándar de los países de la OTAN, estas especificaciones se aplican para la fabricación de las municiones 7.62x51mm y 5.56x45mm.

En Bolivia la encargada de la distribución de municiones a las unidades militares, es la Fábrica Boliviana de Munición (FBM). Los costos a la fecha de realización de este perfil de trabajo de grado son los siguientes. Los cartuchos 7.62x51mm tienen un precio de 5,14 Bs por unidad, y en el caso de los cartuchos 5.56x45mm tienen un precio de 5 Bs por unidad según la información de la FBM.

Debido al costo logístico de la operación la cantidad de entrenamientos de tiro anuales son programados 2 veces al año. Siendo realizados en los exámenes del primer periodo de entrenamiento básico militar individual.

En cuanto a los simuladores de tiro, en Bolivia se realizó una tesis denominada “HERRAMIENTA DE ENTRENAMIENTO DE POLÍGONOS DE TIRO PARA SOLDADOS CON FUSIL EMPLEANDO REALIDAD VIRTUAL” en la EMI CBBA por la Ing. Diana Arleth Muriel Soto y el Ing. Humberto Enrique Pardo Iriarte. Brindando como base de su trabajo realizado para la mejora y continuidad de este, Tomando algunas de las recomendaciones dadas se tiene:

- Generar más escenarios geográficos acompañados de un módulo de clima aleatorio por zonas para generar un entrenamiento con mayor dificultad aumentado la realidad.
- Aumentar un escenario de entrenamiento en el cual se pueda practicar de tiro de acción y reacción, implementando el movimiento del soldado dentro del sistema con sensores infrarrojos para controlar hacia dónde se dirige y no solo enfrentarlos en un puesto fijo.
- Contar con maquetas establecidas de sus armas o futuras armas para la implantación del fusil de hardware a las mismas e introducirlas al sistema haciéndolo más robusto permitiendo el entrenamiento con las antes mencionadas.

Teniendo en cuenta todo lo mencionado anteriormente, lo que se pretende realizar es un simulador para el entrenamiento de tiro con fusiles de asalto (Fal FN, Galil AR, AK-47).

1.3. PLANTEAMIENTO DEL PROBLEMA.

1.3.1. Identificación del problema.

El entrenamiento de tropas bolivianas debido a la limitada cantidad de presupuesto para los ejercicios militares, realizan sus entrenamientos de tiro normalmente en las

revistas(examen) siendo esa su primera experiencia disparando un arma, este entrenamiento supone un costo elevado en la parte logística y de instrumentación, también, limitados a las zonas donde están ubicados los polígonos de tiro, donde la eficiencia combativa del oficial, suboficial y soldado se ve afectada por la falta de práctica.

1.3.1.1. Identificación de la situación problemática.

- La realización de prácticas de tiro supone un costo elevado lo que provoca una disminución de entrenamientos prácticos de tiro durante el año.
- La cantidad limitada de entrenamientos prácticos de tiro realizados por parte de oficiales, suboficiales, sargentos y soldados provoca la reducción en la precisión de tiro.
- Al ejecutar las prácticas en los polígonos o campos de tiro, se genera una costumbre al entorno que provoca falla en el disparo en diferentes entornos de desempeño.
- El Uso de como máximo un tipo de arma en el entrenamiento de tiro con fusiles de asalto provoca que el soldado no dispare las otras dos armas estandarizadas por su división.

1.3.1.2. Identificación de la causa.

- Disminución de entrenamientos prácticos de tiro durante el año.
- Reducción en la precisión de tiro por falta de entrenamiento.
- Precisión del combatiente disminuye en otro ambiente diferente al de costumbre.
- Un soldado no dispara con más de dos armas al año en prácticas de tiro con fusil de asalto.

1.3.2. Formulación del problema.

Las prácticas de tiro se realizan una vez cada semestre y en un solo polígono de tiro demarcado en un área determinada provocan que se delimite la cantidad de entrenamientos debido a los costos elevados por cada operación de entrenamiento de tiro con fusiles de asalto.

1.4. Objetivos y acciones.

Esta sección abarca la descripción de los objetivos y acciones identificadas:

1.4.1. Objetivo general.

Desarrollar un simulador para el entrenamiento de tiro con fusiles de asalto FN FAL, GALIL AR y AK-47, bajo entornos dinámicos y blancos móviles usando realidad virtual para Incrementar la cantidad de entrenamientos de tiro con fusiles de asalto y reducir los costos de entrenamiento del BATALLÓN DE POLICÍA MILITAR III ESTEBAN ARCE”.

1.4.2. Objetivos específicos y acciones de la investigación.

Para el desarrollo de un simulador para el entrenamiento de tiro con fusiles bajo entornos dinámicos y enemigos figurados usando realidad virtual, se tienen los siguientes objetivos específicos:

- Analizar el proceso actual de entrenamiento con fusil y su entorno de desempeño.
- Aplicar el método FDD (Feature-driven Development) para desarrollar el simulador según la arquitectura HLA (High Level Architecture).
- Implementar los entornos dinámicos de realidad virtual en tiempo real a través de los métodos procedurales.
- Establecer las características del disparo en los climas diversos de acuerdo a los terrenos dinámicos.
- Implementar una tabla estadística de tiro para observar la evolución de usuario.
- Modelar en 3d los fusiles PROPUESTOS (FN FAL, GALIL AR Y AK-47).
- Validar el sistema mediante un plan de Quality Assurance (QA).

Tabla 1: Objetivos específicos y acciones

OBJETIVOS ESPECIFICOS	ACCIONES
<p>ANALIZAR EL PROCESO ACTUAL DE ENTRENAMIENTO CON FUSIL Y SU ENTORNO DE DESEMPEÑO.</p>	<ul style="list-style-type: none"> • Recopilar información del proceso de entrenamiento de tiro con fusil en el ejército boliviano. • Reunir información sobre los fusiles utilizados en los entrenamientos de tiro. • Recabar información sobre los diferentes entornos geográficos donde se realizan los entrenamientos de tiro con fusil.
<p>APLICAR EL MÉTODO FDD (FEATURE-DRIVEN DEVELOPMENT) PARA DESARROLLAR EL SIMULADOR SEGÚN LA ARQUITECTURA HLA (HIGH LEVEL ARCHITECTURE).</p>	<ul style="list-style-type: none"> • Efectuar mediante un diagrama de objetos el funcionamiento del simulador para el entrenamiento de tiro con fusiles. • Construir una lista de características del simulador para el entrenamiento de tiro con fusiles. • Agrupar las características del simulador para el entrenamiento de tiro con fusiles en áreas subjetivas. • Planificar en base de cada característica del simulador para el entrenamiento de tiro con fusiles. • Diseñar en base de cada característica del simulador para el entrenamiento de tiro

	con fusiles.
IMPLEMENTAR LOS ENTORNOS DINÁMICOS DE REALIDAD VIRTUAL EN TIEMPO REAL A TRAVÉS DE LOS MÉTODOS PROCEDURALES.	<ul style="list-style-type: none"> • Obtener las curvaturas del entorno mediante el GDEM (Global Digital Elevation Map) para el relevamiento del terreno en base a coordenadas de latitud y longitud. • Codificar el método procedural con las imágenes obtenidas mediante las coordenadas para la generación de terrenos en tiempo real. • Verificar el funcionamiento del método procedural y su funcionalidad a la hora de generar el terreno en tiempo real.

<p>ESTABLECER LAS CARACTERÍSTICAS DEL DISPARO EN LOS CLIMAS DIVERSOS DE ACUERDO A LOS TERRENOS DINÁMICOS</p>	<ul style="list-style-type: none"> • Recopilar información de las variables climatológicas. • Codificar los atributos del clima relacionado al entorno. • Codificar la física relacionada a la trayectoria de disparo.
<p>IMPLEMENTAR UNA TABLA ESTADISTICA DE TIRO PARA OBSERVAR LA EVOLUCIÓN DEL USUARIO</p>	<ul style="list-style-type: none"> • Diseñar una base de datos (SQL) para el almacenamiento de datos estadísticos por usuario. • Crear la base de datos (SQL) • Establecer las tablas y relaciones de la base de datos. • Definir los parámetros estadísticos a ser tomados en cuenta para la tabla. • Codificar la tabla estadística en base al usuario y los parámetros dados anteriormente. • Generar reportes por usuario.
<p>MODELAR EN 3D LOS FUSILES PROPUESTOS (FN FAL, GALIL AR Y AK-47).</p>	<ul style="list-style-type: none"> • Diseñar los fusiles FN FAL, GALIL AR y AK-47 en 3D. • Aplicar las texturas respectivas en los modelos 3D de los fusiles. • Codificar el funcionamiento de las animaciones de los modelos 3D.

<p>VALIDAR EL SISTEMA MEDIANTE UN PLAN DE QUALITY ASSURANCE (QA)</p>	<ul style="list-style-type: none"> • Realizar pruebas de testeo unitario al simulador. • Efectuar pruebas de estrés al simulador. • Realizar el test de integración a los módulos probados en el testeo unitario.
---	--

Fuente: Elaboración propia, 2020.

1.5. JUSTIFICACIÓN

1.5.1. Justificación técnica.

La simulación en paralelo a la realidad virtual, permitirán al usuario poder experimentar una inmersión, generando un mundo virtual a partir de coordenadas obtenidas mediante API'S de satélites estadounidenses y japoneses, lo que ayudará a crear el terreno de manera dinámica.

Para el equipo de realidad virtual, se optará el "Oculus Rift" el cual nos permite desarrollar mediante su SDK brindándonos facilidades para utilizarlos en el IDE, y crear componentes de una manera más sencilla dentro de Unity. En cuanto al hardware se usarán microcontroladores (Arduino) y un mecanismo eléctrico para el control en el retroceso del fusil.

Con Unity se generará los ejecutables para Windows, en una máquina que tenga una tarjeta gráfica mínimamente de la serie 10 de GTX por el hecho de que dicha serie está optimizada para la ejecución de Simuladores o juegos de realidad virtual.

1.5.2. Justificación económica.

El entrenador de tiro con fusiles reducirá los costos a largo plazo de:

- Municionamiento.
- Transporte.

1.5.3. Justificación operativa.

El entrenador de tiro con fusiles ayudará a la capacitación del personal del ejército boliviano, para la aplicación práctica de los fusiles con los que cuenta el ejército de Bolivia.

Al aumentar el número de prácticas de tiro, el personal del ejército de Bolivia aumentará su eficiencia combativa.

1.6. DELIMITACIÓN DE LA INVESTIGACIÓN.

1.6.1. Delimitación temática.

La siguiente tesis de grado tiene como finalidad “Desarrollar un simulador para el entrenamiento de tiro con fusiles de asalto FN FAL, GALIL AR y AK-47, bajo entornos dinámicos y blancos móviles usando realidad virtual. Basándose en una arquitectura HLA (High Level Architecture) y haciendo uso de API's de satélites estadounidenses y japoneses (ASTER GDEM) para la generación de los terrenos por medio de métodos procedurales, modelamiento de objetos en 3D, lenguajes de programación, paradigmas de programación, estructuras de datos, bases de datos, teorías físicas.

En cuanto a la generación de los terrenos dinámicos sólo se tomarán en cuenta las variables climatológicas de los “valles”, y solo se interactuará con el Oculus Rift como parte de hardware.

En cuanto a la arquitectura HLA sólo se utilizará la estructura que conlleva esta arquitectura como una guía.

1.6.2. Delimitación espacial.

El Desarrollo de un simulador para el entrenamiento de tiro con fusiles de asalto FN FAL, GALIL AR y AK-47, bajo entornos dinámicos y blancos móviles usando realidad virtual, estará destinado al “BATALLÓN DE POLICÍA MILITAR III ESTEBAN ARCE”.

1.6.3. Delimitación temporal.

El periodo comprendido de la ejecución de esta tesis de grado está estipulado para la gestión 2020.

1.7. HIPÓTESIS.

El Desarrollo de un simulador para el entrenamiento de tiro con fusiles de asalto FN FAL, GALIL AR y AK-47, bajo entornos dinámicos y blancos móviles usando realidad virtual permitirá realizar mayor cantidad de prácticas de tiro con diferentes fusiles reduciendo los costos de los entrenamientos de tiro con fusiles de asalto del BATALLÓN DE POLICÍA MILITAR III ESTEBAN ARCE”.

1.7.1. Identificación de las variables.

1.7.1.1. Variable independiente.

Simulador para el entrenamiento de tiro con fusiles de asalto FN FAL, GALIL AR y AK-47, bajo entornos dinámicos y blancos móviles usando realidad.

1.7.1.2. Variable dependiente.

- Costo de la práctica de tiro con fusiles.

1.7.2. Definición conceptual de las variables.

1.7.2.1. Variables independientes.

El desarrollo de un simulador para el entrenamiento de tiro con fusiles bajo entornos dinámicos y enemigos figurados utilizando realidad virtual permitirá realizar mayor cantidad de prácticas de tiro con diferentes fusiles reduciendo los costos.

1.7.2.2. Variables dependientes.

- Costo de la práctica de tiro con fusiles de asalto.

El costo del actual proceso de entrenamiento se reducirá de manera notable, debido a que se economizan las municiones y recursos humanos utilizados por entrenamiento.

- Cantidad de entrenamientos por año.

La cantidad de entrenamientos de tiro actuales se incrementarán debido al simulador que se implementara.

1.7.3. Operativización de las variables.

Tabla 2: Operativización de variables.

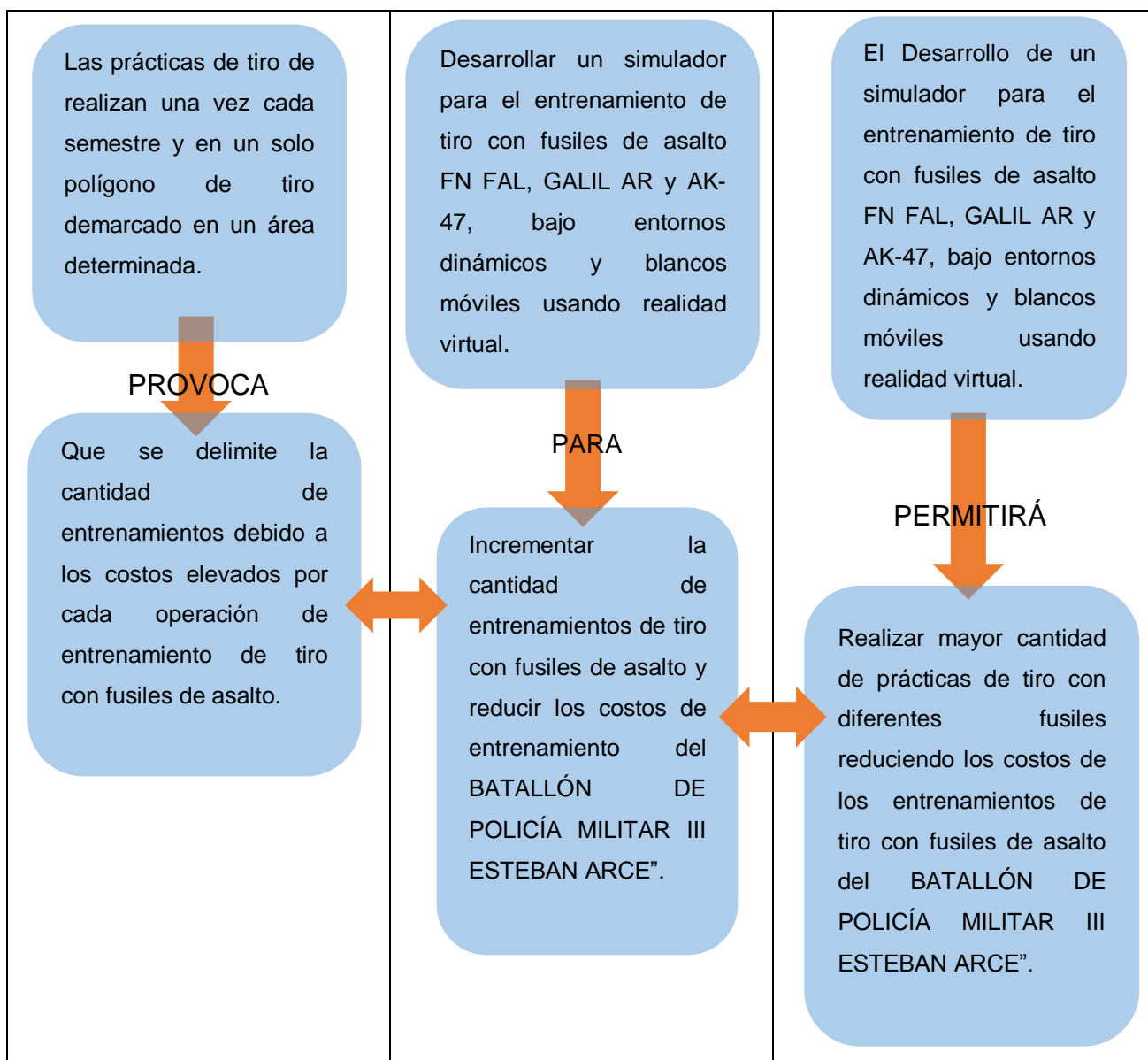
VARIABLE	DIMENSIÓN	INDICADOR
<u>Variable independiente</u> <ul style="list-style-type: none">• Simulador para el entrenamiento de tiro con fusiles de asalto FN FAL, GALIL AR y AK-47, bajo entornos dinámicos y blancos móviles usando realidad virtual”	<ul style="list-style-type: none">• Implementación del simulador.	<ul style="list-style-type: none">• Cantidad de entrenamientos de tiro con fusiles de asalto
<u>Variable dependiente</u> <ul style="list-style-type: none">• Costo de la práctica de tiro con fusiles	<ul style="list-style-type: none">• Recursos utilizados	<ul style="list-style-type: none">• Bs. / Operación.

Fuente: Elaboración propia, 2020.

1.8. MATRIZ DE CONSISTENCIA.

Tabla 3: Matriz de consistencia.

PROBLEMA	OBJETIVO	HIPÓTESIS
----------	----------	-----------



Fuente: Elaboración propia, 2020.

CAPÍTULO II: MARCO TEÓRICO



CAPITULO 2: MARCO TEÓRICO

2.1. ESQUEMA DEL MARCO TEÓRICO

Tabla 4:Esquema del marco teórico.

OBJETIVOS ESPECÍFICOS	ACCIONES	CONTENIDO TEMATICO
Analizar el proceso actual de entrenamiento con fusil y su entorno de desempeño.	<ul style="list-style-type: none"> • Recopilar información del proceso de entrenamiento de tiro con fusil en el ejército boliviano. • Reunir información sobre los fusiles utilizados en los entrenamientos de tiro. • Recabar información sobre los diferentes entornos geográficos donde se realizan los entrenamientos de tiro con fusil. 	Herramienta de entrenamiento de polígonos de tiro para soldados con fusil empleando realidad virtual (Soto Diana).
Aplicar el método FDD (Feature-Driven Development) para desarrollar el simulador	<ul style="list-style-type: none"> • Efectuar mediante un diagrama de objetos el funcionamiento del 	<ul style="list-style-type: none"> - INGENIERÍA DE SOFTWARE. <ul style="list-style-type: none"> ○ Metodologías Ágiles. ○ Feature Driven Development (FDD). ○ Arquitecturas de software. ○ Modelos de objetos. ○ Grupos de servicio.

<p>según la arquitectura HLA (High Level Architecture).</p>	<p>simulador para el entrenamiento de tiro con fusiles.</p> <ul style="list-style-type: none"> • Construir una lista de características del simulador para el entrenamiento de tiro con fusiles. • Agrupar las características del simulador para el entrenamiento de tiro con fusiles en áreas subjetivas. • Planificar en base de cada característica del simulador para el entrenamiento de tiro con fusiles. • Diseñar en base de cada característica del simulador para el entrenamiento de tiro con fusiles. • Componer en base de cada característica del simulador para el entrenamiento de tiro con fusiles. 	<ul style="list-style-type: none"> ○ Reglas. ○ Implementación independencia. ○ Relación de HLA y conceptos orientados a objetos (OO). ○ HLA DEVELOPMENT PROCESS. ○ Reglas de HLA. <p>- SISTEMAS DISTRIBUIDOS.</p> <ul style="list-style-type: none"> ○ SISD ○ MISD ○ SIMD ○ MIMD ○ Memoria compartida
---	--	---

<p>Implementar los entornos dinámicos de realidad virtual en tiempo real a través de los métodos procedurales.</p>	<ul style="list-style-type: none"> • Obtener las curvaturas del entorno mediante el GDEM (Global Digital Elevation Map) para el relevamiento del terreno en base a coordenadas de latitud y longitud. • Codificar el método procedural con las imágenes obtenidas mediante las coordenadas para la generación de terrenos en tiempo real. • Verificar el funcionamiento o del método procedural y su funcionalidad a la hora de generar el terreno en tiempo real. 	<p>REALIDAD VIRTUAL</p> <ul style="list-style-type: none"> - Visión Panorámica. <ul style="list-style-type: none"> ○ Hardware. ○ Software. - Rastreo. <ul style="list-style-type: none"> ○ Seguimiento de la orientación 3d. ○ Posición de seguimiento y orientación. ○ Seguimiento de cuerpos adjuntos. - Interacción. <ul style="list-style-type: none"> ○ Locomoción. ○ Manipulación. - Evaluación de sistemas de realidad virtual y experiencias. <ul style="list-style-type: none"> ○ Entrenamiento perceptivo. ○ Recomendaciones para desarrolladores. ○ Comodidad y enfermedad de realidad virtual. - Métodos Procedurales. <ul style="list-style-type: none"> ○ Generación del terreno base. <ul style="list-style-type: none"> ▪ Método "Pink Noise". ▪ Diagramas de voronoi. ▪ Combinación y perturbación. <p>UNITY.</p> <ul style="list-style-type: none"> - Importado. <ul style="list-style-type: none"> ○ Importando modelos. - Entradas. <ul style="list-style-type: none"> ○ Entrada unity xr. - Físicas. <ul style="list-style-type: none"> ○ Físicas 3D. - Scripting. <ul style="list-style-type: none"> ○ C#. - Audio. - Animación. <ul style="list-style-type: none"> ○ Animator Controllers. - XR.
<p>Establecer las características del disparo en</p>	<ul style="list-style-type: none"> • Recopilar información de las variables climatológicas. 	<p>FÍSICA</p> <ul style="list-style-type: none"> - Cinemática <ul style="list-style-type: none"> ○ Tiro Parabólico.

<p>los climas diversos de acuerdo a los terrenos dinámicos.</p>	<ul style="list-style-type: none"> • Codificar los atributos del clima relacionado al entorno. • Codificar la física relacionada a la trayectoria de disparo. • Codificar la física relacionada a la trayectoria del disparo. 	<ul style="list-style-type: none"> - Leyes de newton <ul style="list-style-type: none"> ○ Ley de la inercia ○ Segunda ley de Newton ○ Tercera ley de Newton ○ Momento Lineal ○ Conservación del momento lineal <p>UNITY.</p> <ul style="list-style-type: none"> - Importado. <ul style="list-style-type: none"> ○ Importando modelos. - Entradas. <ul style="list-style-type: none"> ○ Entrada unity xr. - Físicas. <ul style="list-style-type: none"> ○ Físicas 3D. - Scripting. <ul style="list-style-type: none"> ○ C#. - Audio. - Animación. <ul style="list-style-type: none"> ○ Animator Controllers. - XR.
<p>Implementar una tabla estadística de tiro para observar la evolución del usuario</p>	<ul style="list-style-type: none"> • Diseñar una base de datos (SQL) para el almacenamien to de datos estadísticos por usuario. • Crear la base de datos (SQL). • Establecer las tablas y relaciones de la base de datos. • Definir los parámetros estadísticos a ser tomados en cuenta para la tabla. • Codificar la tabla estadística en 	<ul style="list-style-type: none"> - BASES DE DATOS <ul style="list-style-type: none"> ○ Creación de tablas en SQL ○ Especificación de los tipos de datos en una columna

	<p>base al usuario y los parámetros dados anteriormente.</p> <ul style="list-style-type: none"> • Generar reportes por usuario. 	
Modelar en 3D los fusiles propuestos (FN FAL, GALIL AR Y AK-47).	<ul style="list-style-type: none"> • Diseñar los fusiles FN FAL, GALIL AR y AK-47 en 3D. • Aplicar las texturas respectivas en los modelos 3D de los fusiles. • Codificar el funcionamiento de las animaciones de los modelos 3D. 	<ul style="list-style-type: none"> - MODELADO 3D CON BLENDER. <ul style="list-style-type: none"> ○ Bloqueando. ○ Horneado de texturas. ○ Materiales.
Validar el sistema mediante un plan de Quality Assurance (QA).	<ul style="list-style-type: none"> • Realizar pruebas de testeo unitario al simulador. • Efectuar pruebas de estrés al simulador. • Realizar el test de integración a los módulos probados en el testeo unitario. 	<p>INGENIERÍA DE SOFTWARE</p> <ul style="list-style-type: none"> - Quality Assurance <ul style="list-style-type: none"> ○ Fundamentos de calidad de software ○ Calidad del proceso de software ○ Verificación y validación ○ Inspecciones ○ Técnicas estáticas ○ Técnicas Dinámicas ○ Testing

Fuente: Elaboración propia, 2020.

2.2. REALIDAD VIRTUAL

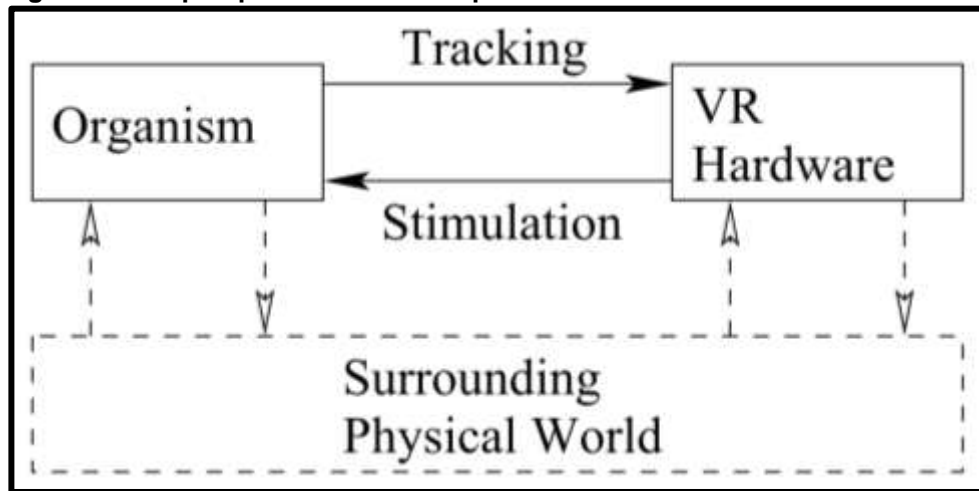
La realidad virtual (VR) es una tecnología poderosa que promete cambiar nuestras vidas como ninguna otra. Al estimular artificialmente nuestros sentidos, nuestros cuerpos se engañan para aceptar otra versión de la realidad. La realidad virtual es como un sueño despierto que podría tener lugar en un mundo mágico de dibujos animados, o podría transportarnos a otra parte de la Tierra o el universo. Es el siguiente paso en un camino que incluye muchos medios familiares, desde pinturas hasta películas y videojuegos. Incluso podemos socializar con personas dentro de nuevos mundos, que pueden ser reales o artificiales. (LaValle, 2019, pp: 3)

2.2.1. Hardware.

El primer paso para comprender cómo funciona la realidad virtual es considerar qué constituye todo el sistema de realidad virtual. Es tentador pensar que se trata simplemente de los componentes de hardware, como computadoras, auriculares y controladores.

El hardware produce estímulos que anulan los sentidos del usuario. El hardware de realidad virtual logra esto mediante el uso de sus propios sensores, lo que permite seguir los movimientos del usuario. El seguimiento de la cabeza es lo más importante, pero el seguimiento también puede incluir presionar botones, movimientos del controlador, movimientos oculares, o los movimientos de cualquier otra parte del cuerpo. Finalmente, también es importante considerar el mundo físico circundante como parte del sistema VR. A pesar de la estimulación proporcionada por el hardware VR, el usuario siempre tendrá otros sentidos que responden a estímulos del mundo real. Ella también tiene la habilidad para cambiar su entorno a través de movimientos corporales. El hardware VR también podría rastrear objetos que no sean el usuario, especialmente si la interacción con ellos es parte del Experiencia de realidad virtual. A través de una interfaz robótica, el hardware VR también puede cambiar el mundo real. Un ejemplo es la tele operación de un robot a través de una interfaz VR.

Figura 1: Una perspectiva en tercera persona de un sistema de realidad virtual



Fuente: LaValle, 2019

Es incorrecto suponer que el hardware y el software diseñados son el sistema VR completo: el organismo y su interacción con el hardware son igualmente importantes. Además, las interacciones con el mundo físico circundante continúan ocurriendo durante una realidad virtual experiencial. (LaValle, 2019, pp: 39-40)

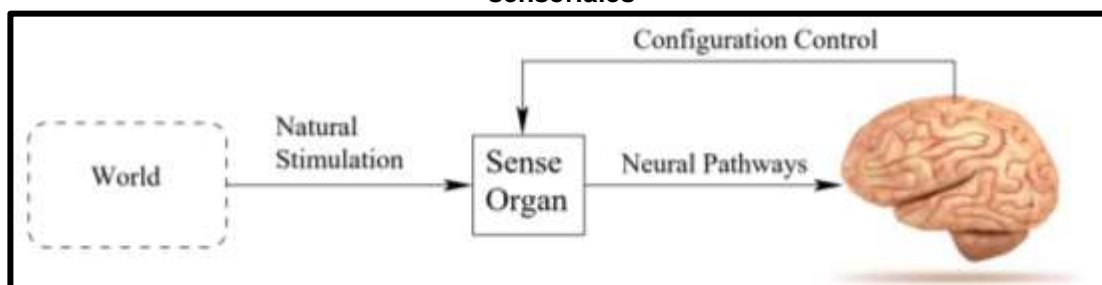
2.2.1.1. Sensores y órganos sensoriales.

¿Cómo se extrae la información de lo físico? ¿mundo? Claramente, esto es crucial para un sistema de realidad virtual. En ingeniería, un transductor se refiere a un dispositivo que convierte la energía de una forma a otra. Un sensor es un especial transductor que convierte la energía que recibe en una señal para un circuito eléctrico. Esta puede ser una señal analógica o digital, dependiendo del tipo de circuito. Un sensor normalmente tiene un receptor que recolecta la energía para la conversión. Los organismos trabajan de manera similar. El "sensor" se llama órgano sensorial, siendo ejemplos comunes Ojos y oídos. Debido a que nuestros "circuitos" se forman a partir de neuronas interconectadas, Los órganos sensoriales convierten la energía en impulsos neuronales. Esto no debería ser sorprendente porque nosotros y nuestros ingenieros los dispositivos compartimos el mismo mundo físico: las leyes de la física y la química permanecen lo mismo. (LaValle, 2019, pp: 40)

2.2.1.2. Espacio de configuración de los órganos sensoriales.

A medida que el usuario se mueve a través de lo físico mundo, sus órganos sensoriales se mueven junto con él. Además, algunos órganos sensoriales moverse en relación con el esqueleto del cuerpo, como nuestros ojos que giran dentro de sus cuencas. Cada órgano sensorial tiene un espacio de configuración, que corresponde a todas las formas posibles Se puede transformar o configurar. El aspecto más importante de esto es la cantidad de grados de libertad o DOF del órgano sensorial. Tenga en cuenta que un objeto rígido que se mueve a través de lo ordinario. El espacio tiene seis DOF (Degrees of Freedom o Grados de Libertad). Tres DOF corresponden a su posición cambiante en el espacio: 1) movimiento de lado a lado, 2) movimiento vertical y 3) movimiento más cercano-más alejado.

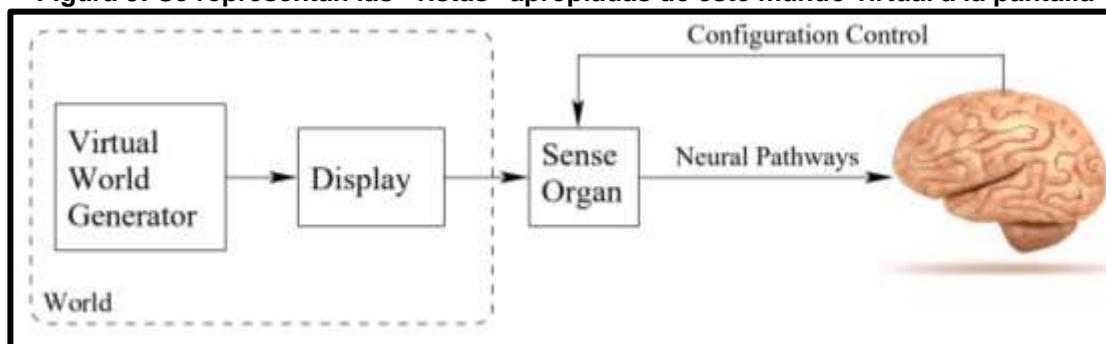
Figura 2: El cerebro (y las partes del cuerpo) controlan la configuración de los órganos sensoriales



Fuente: LaValle, 2019

La Figura 3 en comparación con la Figura 2, muestra un sistema de realidad virtual "secuestra" cada sentido por reemplazando la estimulación natural con estimulación artificial que es proporcionada por hardware llamado una pantalla. Usando una computadora, un generador de mundo virtual mantiene Un mundo coherente y virtual. Se representan las "vistas" apropiadas de este mundo virtual a la pantalla.

Figura 3: Se representan las "vistas" apropiadas de este mundo virtual a la pantalla

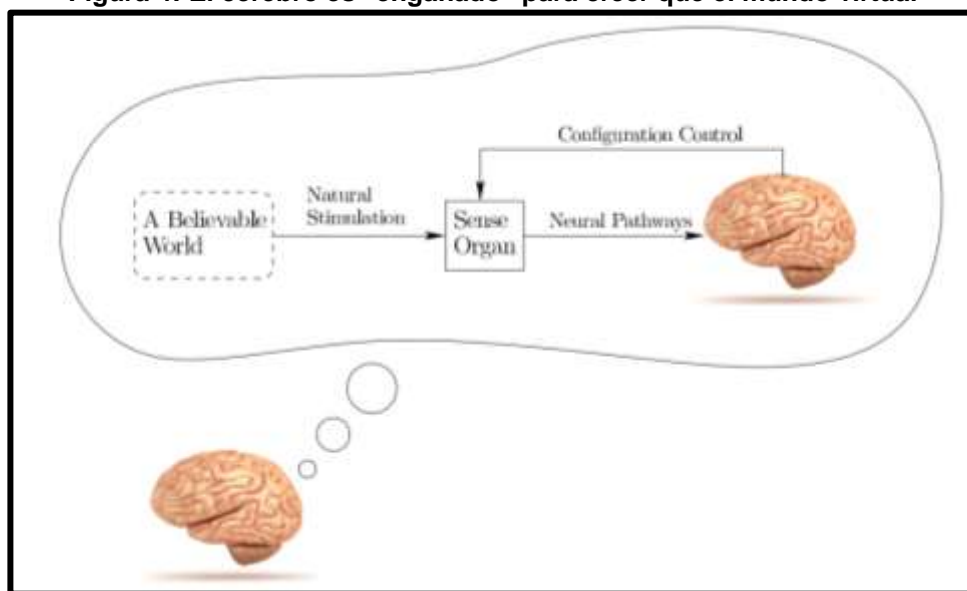


Fuente: LaValle, 2019

Tres DOF corresponden a posibles formas en que el objeto podría rotarse; en otra palabra, exactamente tres parámetros independientes son necesarios para especificar cómo el objeto está orientado. Estos se denominan guiñada, cabeceo y balanceo. Como ejemplo, considere su oreja izquierda. Mientras gira su cabeza o mueve su cuerpo a través del espacio, cambia la posición del oído, así como su orientación. Esto produce seis DOF. Lo mismo es cierto para su ojo derecho, pero también es capaz de girando independientemente de la cabeza. Ten en cuenta que nuestros cuerpos tienen muchos más grados de libertad, que afectan la configuración de nuestros órganos sensoriales. Un seguimiento el sistema puede ser necesario para determinar la posición y orientación de cada sentido órgano que recibe estímulos artificiales, que se explicarán en breve.

Una vista abstracta de la Figura 2 ilustra el funcionamiento normal de uno de nuestros sentidos órganos sin interferencia del hardware VR. El cerebro controla su configuración, mientras que el órgano sensorial convierte la estimulación natural del ambiente en impulsos neuronales que se envían al cerebro. La Figura 3 muestra cómo aparece en un Sistema de realidad virtual. El hardware VR contiene varios componentes que serán discutidos.

Figura 4: El cerebro es "engañado" para creer que el mundo virtual



Fuente: LaValle, 2019

La Figura 4 muestra que, si se hace bien, el cerebro es "engañado" para creer que el mundo virtual es, de hecho, el mundo físico circundante y la estimulación natural es el resultado de eso.

Dentro de poco, un Virtual World Generator (VWG) se ejecuta en una computadora y produce "Otro mundo", que podría ser muchas posibilidades, como una simulación pura de un mundo sintético, una grabación del mundo real o una conexión en vivo a otra parte del mundo real. El humano percibe el mundo virtual a través de cada objetivo. Órgano sensorial usando una pantalla, que emite energía diseñada específicamente para imitar el tipo de estímulo que aparecería sin VR. El proceso de convertir información del VWG en salida para la pantalla se denomina renderizado. En el caso de los ojos humanos, la pantalla puede ser la pantalla de un teléfono inteligente o la pantalla de un video proyector. En el caso de los oídos, la pantalla se denomina altavoz. (Una pantalla no necesita ser visual, aunque este es el uso común en todos los días vida.) Si el sistema VR es efectivo, entonces el cerebro es "engañado" en el sentido se muestra en la Figura 4. El usuario debe creer que la estimulación de los sentidos. es natural y proviene de un mundo plausible, siendo consistente con al menos algunas experiencias pasadas. (LaValle, 2019, pp: 40-43)

2.2.1.3. Visual: fijo en el mundo vs. fijado por el usuario.

Ahora considere agregar una pantalla visual. Puede que no te preocupes demasiado por la ubicación percibida de artistas e instrumentos mientras escucha música, pero notará rápidamente si sus ubicaciones no aparecen correcto a tus ojos. Nuestro sentido de la visión es mucho más poderoso y complejo que Nuestro sentido del oído. La Figura 5 (a) muestra un sistema CAVE (cueva), que es paralelo al sistema de sonido envolvente de muchas maneras. El usuario nuevamente se sienta en el centro mientras Las pantallas alrededor de la periferia presentan estímulos visuales en sus ojos. Los altavoces son reemplazados por pantallas de video. La Figura 5 (b) muestra a un usuario usando un auricular VR, que es paralelo a los auriculares. Supongamos que la pantalla frente a los ojos del usuario muestra una imagen fija en el auricular. Si el usuario gira la cabeza, la imagen se percibirá como adjunta a la cabeza. Esto ocurriría, por ejemplo, si gira la cabeza mientras usa Viewmaster. Si en

cambio quisieras percibir la imagen como parte de un mundo fijo a tu alrededor, entonces la imagen dentro del auricular debe cambiar para compensar a medida que gira la cabeza. El mundo virtual circundante debería ser contra rotado. Una vez que aceptamos que tales transformaciones son necesarias, se convierte en un importante desafío de ingeniería para estimar la cantidad de movimiento de cabeza y ojos que ha ocurrido y aplica la transformación apropiada de manera oportuna y precisa.

Figura 5: (a) Sistema cave. (b) Auricular VR.



Fuente: LaValle, 2019

Si esto no se maneja bien, entonces los usuarios podrían tener problemas o ser poco convincentes. experiencias. Peor aún, podrían ser víctimas de la enfermedad de realidad virtual. Este es uno de las principales razones por las que la popularidad de los auriculares VR disminuyó en la década de 1990. El componente La tecnología aún no era lo suficientemente buena. Afortunadamente, la situación ha mejorado mucho. en el presente. Para el audio, pocos parecían molestarse con esta transformación, pero para la contraparte visual, es absolutamente crítico. Una nota final es que el seguimiento y aplicar transformaciones también se hace necesario en los sistemas CAVE si queremos que las imágenes en las pantallas que se modificarán de acuerdo con los cambios en las posiciones de los ojos en el interior de la habitación.

Ahora que tiene una comprensión de alto nivel de las disposiciones de hardware comunes, analizaremos más de cerca los componentes de hardware que son ampliamente disponible para construir sistemas de realidad virtual. Se espera que

estos cambien rápidamente, con Disminución de costos y mejora del rendimiento. También esperamos que muchos dispositivos nuevos aparezcan en el mercado en los próximos años. El conocimiento de la tecnología actual proporciona ejemplos concretos para aclarar los conceptos fundamentales de realidad virtual.

Los componentes de hardware de los sistemas de realidad virtual se clasifican convenientemente como:

- Pantallas (salida): dispositivos que estimulan cada uno un órgano sensorial.
- Sensores (entrada): dispositivos que extraen información del mundo real.
- Computadoras: dispositivos que procesan entradas y salidas secuencialmente.

Figura 6: (a) El Sistema Touch X by 3D systems. (b) Algunos de los controladores de juego que ocasionalmente vibran



Fuente: LaValle, 2019

La Figura 6 muestra dos ejemplos de dispositivos de retroalimentación háptica. (a) El sistema Touch X de 3D Systems el cual permite al usuario sentir una fuerte resistencia al hurgar en un virtual objeto con un lápiz real. Un brazo robot proporciona las fuerzas apropiadas. (b) Algunos los controladores de juego ocasionalmente vibran. (LaValle, 2019, pp: 45-47)

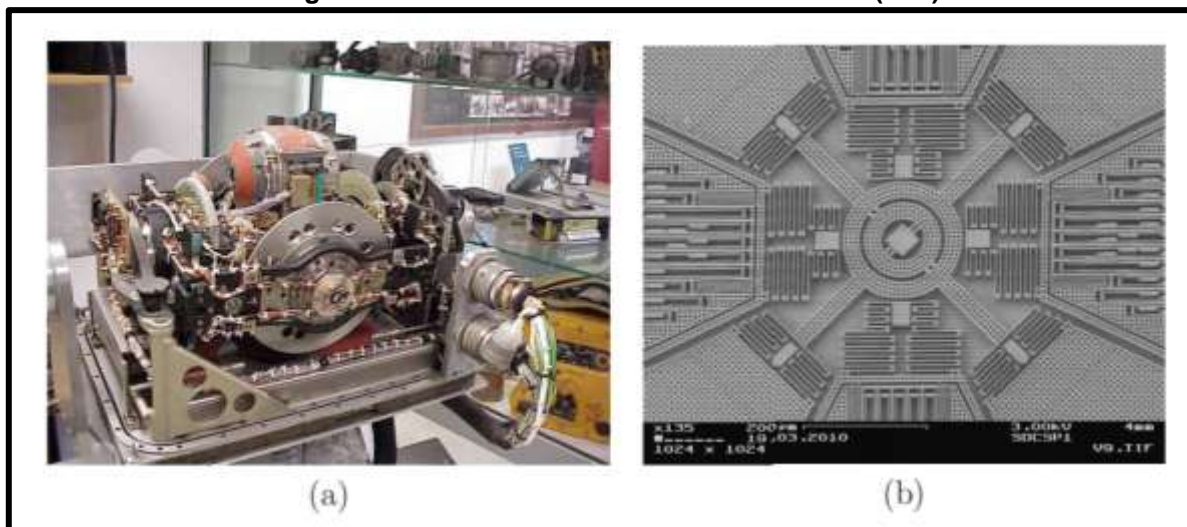
2.2.1.4. Pantallas.

Una pantalla genera estímulos para un órgano sensorial específico. La visión es nuestro sentido dominante, y cualquier pantalla construida para el ojo debe causar la deseada imagen que se formará en la retina. (LaValle, 2019, pp: 46-47)

2.2.1.5. Sensores.

Considere el lado de entrada del hardware VR para visual y pantallas auditivas montadas en el cuerpo, la posición y orientación del órgano sensorial debe ser rastreado por sensores para adaptar adecuadamente el estímulo. La orientación La parte generalmente se logra mediante una unidad de medición inercial o IMU. El principal el componente es un giroscopio, que mide su propia velocidad de rotación; la tasa es se conoce como velocidad angular y tiene tres componentes. Mediciones desde el giroscopio se integra a lo largo del tiempo para obtener una estimación del cambio acumulativo en orientación El error resultante, llamado error de deriva, crecería gradualmente a menos que Se utilizan otros sensores. Para reducir el error de deriva, las IMU también contienen un acelerómetro y posiblemente un magnetómetro.

Figura 7: Las unidades de medición INERCIAL (IMU)

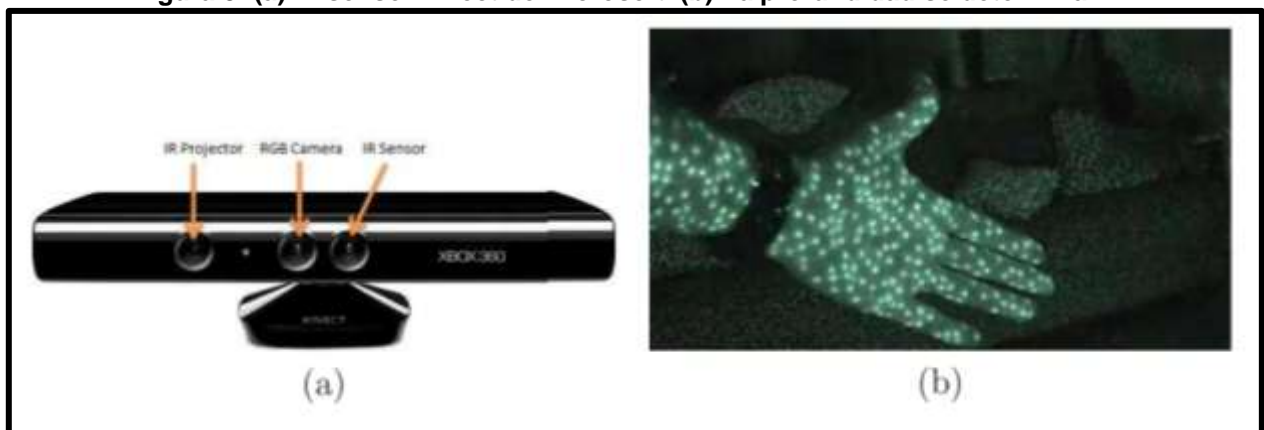


Fuente: LaValle, 2019

La Figura 7 muestra que las unidades de medición inercial (IMU) han pasado de sistemas mecánicos grandes y pesados a circuitos MEMS microscópicos y baratos. (a) La inercia LN-3 Sistema de navegación, desarrollado en la década de 1960 por Litton Industries. (b) El interno estructuras de un giroscopio MEMS, para el cual el ancho total es inferior a 1 mm. Como grandes sistemas mecánicos en aviones y misiles para ser pequeños dispositivos en el interior de teléfonos inteligentes. Debido a su pequeño tamaño, peso y costo, las IMU puede integrarse fácilmente en dispositivos portátiles. Son uno de las más importantes tecnologías habilitadoras

para la generación actual de auriculares VR y son principalmente usado para rastrear la orientación de la cabeza del usuario. Las cámaras digitales proporcionan otra fuente crítica de información para los sistemas de seguimiento. Al igual que las IMU, se han vuelto cada vez más baratas y portátiles debido a la industria de teléfonos inteligentes, al mismo tiempo que mejora la calidad de la imagen. Cámaras habilite enfoques de seguimiento que exploten la visibilidad de la línea de visión. La idea es identificar características o marcadores en la imagen que sirvan como puntos de referencia para un movimiento objeto o un fondo estacionario. Dichas restricciones de visibilidad limitan severamente Posibles posiciones y orientaciones de objetos. Las cámaras estándar forman pasivamente imagen enfocando la luz a través de un sistema óptico, muy parecido al ojo humano. Una vez que se conocen los parámetros de calibración de la cámara, se conoce un marcador observado Acostarse a lo largo de un rayo en el espacio. Las cámaras se usan comúnmente para rastrear ojos, cabezas, manos, cuerpos humanos enteros y cualquier otro objeto en el mundo físico. Uno de los principales desafíos en la actualidad es obtener un rendimiento confiable y preciso sin colocar marcadores especiales en el usuario u objetos alrededor de la escena. A diferencia de las cámaras estándar, las cámaras de profundidad trabajan activamente proyectando luz en la escena y luego observar su reflejo en la imagen. Esto es típicamente hecho en el espectro infrarrojo (IR) para que los humanos no lo noten; Ver Figura 8.

Figura 8: (a) El sensor kinect de microsoft. (b) La profundidad se determina



Fuente: LaValle, 2019

En la Figura 8 se tiene (a) El sensor Kinect de Microsoft reúne una imagen RGB ordinaria y un mapa de profundidad (la distancia del sensor para cada píxel). (b) La profundidad se determina observando las ubicaciones de los puntos IR proyectados en una imagen obtenida de una cámara IR. (LaValle, 2019, pp: 47-49)

2.2.1.6. Computadoras.

Una computadora ejecuta el generador de mundo virtual (VWG). ¿Dónde debería ser esta computadora? Aunque no es importante para pantallas fijas en todo el mundo, en la ubicación es crucial para pantallas fijas al cuerpo. Si se necesita una PC separada para alimentar el sistema, luego se debe proporcionar una comunicación rápida y confiable entre los auriculares y la PC. Esta conexión se realiza actualmente por cables, lo que lleva a un incómodo atar; las velocidades inalámbricas actuales no son suficientes. Como habrás notado, la mayoría de los sensores necesarios existen en un teléfono inteligente, así como en una computadora moderadamente potente. Por lo tanto, un teléfono inteligente se puede colocar en un estuche con lentes para proporcionar una realidad virtual experiencia con pocos costos adicionales (Figura 9). La limitación, sin embargo, es que el VWG debe ser más simple que en el caso de una PC separada para que funcione con hardware informático menos potente. En el futuro cercano, esperamos ver auriculares inalámbricos todo en uno que contengan todas las partes esenciales de los teléfonos inteligentes para entregar Experiencias de realidad virtual. Esto eliminará los componentes innecesarios de los teléfonos inteligentes. (como el caso adicional), y en su lugar tendrá ópticas personalizadas, microchips, y sensores para realidad virtual.

Además de los principales sistemas informáticos, hardware informático especializado puede ser utilizado Las unidades de procesamiento gráfico (GPU) se han optimizado para Representación rápida de gráficos en una pantalla y actualmente se están adaptando a manejar las demandas de rendimiento específicas de la realidad virtual. Además, un chip de interfaz de pantalla convierte un video de entrada en comandos de pantalla. Finalmente, los microcontroladores se utilizan con frecuencia para recopilar información de los dispositivos de detección y enviarlos a la computadora

principal que utiliza protocolos estándar, como USB. Para concluir con hardware, (LaValle, 2019, pp: 49-50)

Figura 9: (a) Google Cardboard. (b) Samsung Gear VR



Fuente: LaValle, 2019

Como se puede observar en la Figura 9 se tiene dos auriculares que crean una experiencia de realidad virtual al soltar un teléfono inteligente en un caso (a) Google Cardboard funciona con una amplia variedad de teléfonos inteligentes. (b) Samsung Gear VR está optimizado para un teléfono inteligente en particular (en este caso, el Samsung S6).

Figura 10: Desmontaje de los auriculares oculus rift dk2



Fuente: LaValle, 2019

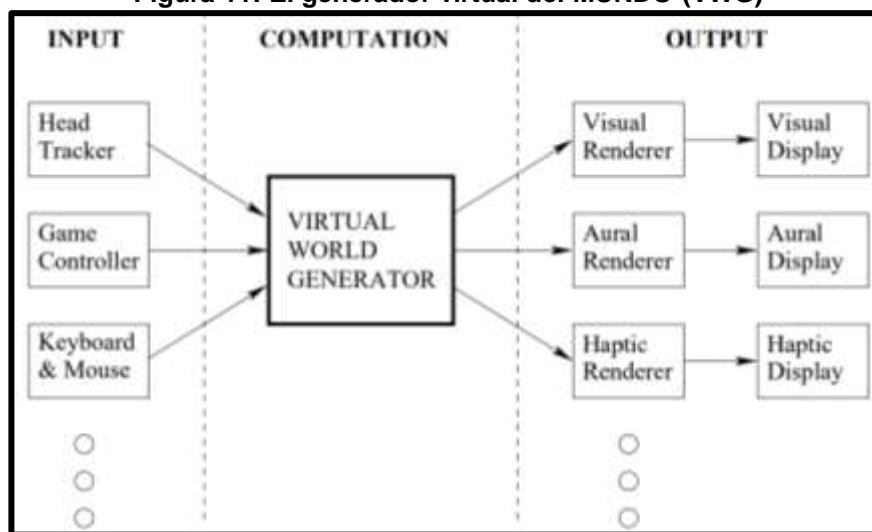
En la Figura 10 podemos observar cada uno de los componentes de hardware que se tiene dentro de un oculus rift.

2.2.2. Software.

Desde el punto de vista de un desarrollador, sería ideal programar el sistema VR por proporcionando descripciones de alto nivel y haciendo que el software determine automáticamente Todos los detalles de bajo nivel. En un mundo perfecto, habría un motor de realidad virtual, que cumple un propósito similar a los motores de juego disponibles hoy para crear video juegos. Si el desarrollador sigue patrones que muchos antes de ella han implementado ya, entonces se pueden evitar muchos detalles complicados simplemente llamando a funciones de una biblioteca de software bien diseñada. Sin embargo, si el desarrollador quiere probar algo original, entonces ella tendría que diseñar las funciones desde cero. Esto requiere una comprensión más profunda de los fundamentos de realidad virtual, a la vez que es familiarizado con las operaciones del sistema de nivel inferior. Desafortunadamente, actualmente estamos muy lejos de tener motores de realidad virtual totalmente funcionales y de propósito general. A medida que se amplían las aplicaciones de VR, los motores de VR especializados son También es probable que surja. Por ejemplo, uno podría ser el objetivo de la cinematografía inmersiva, mientras que otro está orientado al diseño de ingeniería. ¿Qué componentes serán ser más como parte de un "sistema operativo" de realidad virtual y que será más alto componentes de nivel "motor"? Dada la situación actual, los desarrolladores probablemente serán implementando gran parte de la funcionalidad de sus sistemas de realidad virtual desde cero. Esta puede implicar la utilización de un Kit de Desarrollo de Software (SDK) para auriculares particulares que maneja las operaciones de nivel más bajo, como controladores de dispositivos, seguimiento de cabezales, y salida de pantalla. Alternativamente, podrían encontrarse utilizando un motor de juego que se ha adaptado recientemente para la realidad virtual, a pesar de que fue fundamentalmente diseñado para videojuegos en una pantalla. Esto puede evitar un esfuerzo sustancial al principio, pero puede ser engorroso cuando alguien quiere implementar ideas que son no forma parte de los videojuegos estándar. ¿Qué componentes de software se necesitan para producir una experiencia de realidad virtual? presenta una vista de alto nivel que destaca el papel central del generador de mundo virtual (VWG). El VWG recibe entradas de sistemas de bajo nivel que

indican qué está haciendo el usuario en el mundo real. Un rastreador principal proporciona estimaciones oportunas de la posición y orientación de la cabeza del usuario. Teclado, mouse y controlador de juego los eventos llegan en una cola que están listos para ser procesados. El papel clave del VWG es mantener suficiente "realidad" interna para que los renderizadores que puedan extraer información que necesitan para calcular salidas para sus pantallas.

Figura 11: El generador virtual del MUNDO (VWG)



Fuente: LaValle, 2019

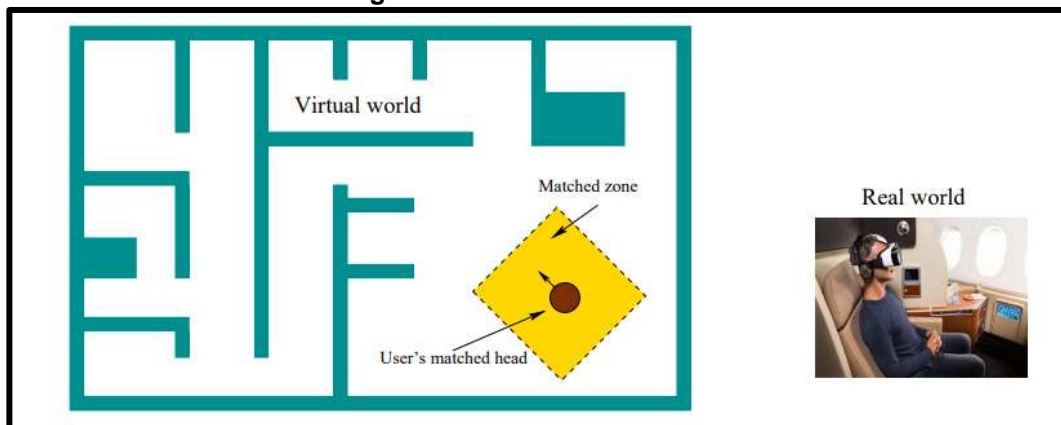
El Virtual World Generator (VWG) mantiene otro mundo, que podría ser sintético, real o alguna combinación. Desde una perspectiva computacional, las entradas se reciben del usuario y su entorno, y vistas apropiadas del mundo se representan en pantallas. (LaValle, 2019, pp: 50)

2.2.2.1. Movimiento emparejado del mundo se representan en pantallas.

La operación más básica del VWG es mantener una correspondencia entre los movimientos del usuario en el mundo real y el mundo virtual; En el mundo real, los movimientos del usuario se limitan a una región segura, que nosotros llamaremos a la zona coincidente. Imagine la zona coincidente como un lugar donde lo real y mundos virtuales perfectamente alineados. Uno de los mayores desafíos es el desajuste de obstáculos: ¿Qué pasa si el usuario está bloqueado en el mundo virtual pero no en el mundo real? ¿mundo? Lo contrario también es posible. En una experiencia sentada, el usuario se sienta en una silla mientras usa auriculares. La zona coincidente en

este caso es una región pequeña, como un metro cúbico, en el que los usuarios pueden mover sus cabezas. Los movimientos de la cabeza deben ser emparejado entre los dos mundos. Si el usuario no está limitado a un asiento, entonces la zona coincidente podría ser una habitación completa o un campo al aire libre. Tenga en cuenta que la seguridad se convierte en un problema porque el usuario puede derramar un trago, golpear paredes o caer en fosas que existen solo en el mundo real, pero que no son visibles en el mundo virtual. Más grande Las zonas coincidentes tienden a generar mayores problemas de seguridad. Los usuarios deben asegurarse de que la zona coincidente está libre de peligros en el mundo real, o el desarrollador debería desarrollarlos visibles en el mundo virtual. Ver Figura 12.

Figura 12: La zona coincidente



Fuente: LaValle, 2019

¿Qué movimientos del mundo real deberían reflejarse en el mundo virtual? Esto varía entre las experiencias de realidad virtual. En un auricular VR que muestra imágenes al ojo, los movimientos de la cabeza deben coincidir para que el renderizador visual use el correcto punto de vista en el mundo virtual. Otras partes del cuerpo son menos críticas, pero pueden ser importantes si el usuario necesita realizar coordinación mano-ojo o mira otras partes de su cuerpo y espera que se muevan naturalmente. (LaValle, 2019, pp: 50-53)

2.2.2.2. Locomoción de usuario.

En muchas experiencias de realidad virtual, los usuarios quieren moverse bien afuera de la zona coincidente. Esto motiva la locomoción, lo que significa moverse en el mundo virtual, mientras que este movimiento no coincide con el mundo real.

Imagínate, desea explorar una ciudad virtual mientras permanece sentado en el mundo real. ¿Cómo debería lograr esto? Podrías abrir un mapa y señalar a dónde quieres ir, con una operación de teletransportación rápida que lo envía al destino. Una famosa opción es moverse en el mundo virtual operando un controlador de juego, mouse o teclado. Al presionar botones o mover perillas, usted mismo en el mundo virtual podría caminar, correr, saltar, nadar, volar, etc. Tú podrías también subir a bordo de un vehículo en el mundo virtual y opere sus controles para moverse tú mismo. (LaValle, 2019, pp: 54)

2.2.2.3. Rastreo.

Hacer un seguimiento del movimiento en el mundo físico es una parte crucial de cualquier sistema de realidad virtual. El seguimiento fue uno de los mayores obstáculos para llevar los auriculares de realidad virtual a la electrónica de consumo, y seguirá siendo un gran desafío debido a nuestro deseo de expandir y mejorar las experiencias de realidad virtual. Los métodos de seguimiento de alta precisión han sido habilitados principalmente por componentes de hardware básicos, como las unidades de medición de inercia (IMU) y las cámaras, que se han desplomado en tamaño y costo debido a la industria de los teléfonos inteligentes. (LaValle, 2019, pp: 249-250)

Pueden aparecer tres categorías de seguimiento en los sistemas de realidad virtual, en función de lo que se está rastreando:

- **Los órganos sensoriales del usuario:** los órganos sensoriales, como los ojos y los oídos, tienen DOF controlados por el cuerpo. Si una pantalla está unida a un órgano sensorial, y se debe percibir que en realidad virtual está unida al mundo circundante, entonces la posición y la orientación del órgano deben rastrearse. El inverso de la transformación rastreada se aplica al estímulo para "deshacer" correctamente estos DOF.
- **Las otras partes del cuerpo del usuario:** si al usuario le gustaría ver un mensaje convincente representación de su cuerpo en el mundo virtual, entonces su movimiento debe ser rastreado para que pueda reproducirse en la zona correspondiente. Quizás se necesiten expresiones faciales o gestos

con las manos para la interacción. Aunque la combinación perfecta es ideal para rastrear órganos sensoriales, no es necesario para rastrear otras partes del cuerpo. Pequeños movimientos en el mundo real podrían convertirse en movimientos de mundo virtual más grandes para que el usuario ejerza menos energía. En el caso límite, el usuario podría simplemente presionar un botón para cambiar la configuración del cuerpo. Por ejemplo, podría agarrar un objeto en su mano virtual con un solo click.

- **El resto del entorno:** en el mundo real que rodea al usuario, los objetos físicos pueden ser rastreados. Para los objetos que existen en el mundo físico, pero no en el mundo virtual, el sistema puede alertar al usuario de su presencia por razones de seguridad. Imagine que el usuario está a punto de golpear una pared o tropezar con un niño pequeño. En algunas aplicaciones de realidad virtual, los objetos físicos rastreados pueden coincidir con la realidad virtual para que el usuario reciba retroalimentación táctil mientras interactúa con ellos. En otras aplicaciones, como la telepresencia, una gran parte del mundo físico podría "introducirse" en el mundo virtual a través de la captura en vivo.

2.2.2.4. Seguimiento de cuerpos adjuntos.

Muchos problemas de seguimiento implican estimar el movimiento de un cuerpo en relación con otro cuerpo en movimiento adjunto. Por ejemplo, un ojo gira dentro de su cuenca, que es parte del cráneo. Aunque el ojo puede tener seis DOF cuando se trata como un cuerpo rígido en el espacio, su posición y orientación se caracterizan suficientemente con dos o tres parámetros una vez que se da la postura de la cabeza. Otros ejemplos incluyen la cabeza en relación con el torso, una mano en relación con la muñeca y la punta de un dedo en relación con su hueso medio. Todo el cuerpo humano puede incluso organizarse en un árbol de cuerpos unidos, basado en un esqueleto. Además, los cuerpos pueden estar unidos de manera similar para otros organismos, como perros o monos, y maquinaria, como robots o automóviles. En el caso de un automóvil, las ruedas giran en relación con el cuerpo. En todos estos casos, el resultado es un sistema multicuerpo. La caracterización matemática de las poses de los cuerpos entre sí se llama cinemática multicuerpo, y la

determinación completa de sus velocidades y aceleraciones se llama dinámica multicuerpo. (LaValle, 2019, pp: 268)

2.2.3. Interacción.

¿Cómo deberían interactuar los usuarios con el mundo virtual? ¿Cómo deberían moverse? ¿Cómo pueden agarrar y colocar objetos? ¿Cómo deberían interactuar con representaciones mutuas? ¿Cómo deberían interactuar con los archivos o Internet? La siguiente información sugiere muchas interfaces posibles.

Principio de simulación universal:

- “Cualquier mecanismo de interacción del mundo real se puede simular en realidad virtual.”

El concepto más importante es la reasignación, en la que un movimiento en el mundo real puede mapearse en un movimiento sustancialmente diferente en el mundo virtual. Esto permite muchos mecanismos de interacción poderosos. La tarea es desarrollar las que sean fáciles de aprender, fáciles de usar, efectivas para la tarea y que brinden una experiencia de usuario cómoda. (LaValle, 2019, pp: 287-288)

Consideraciones de diseño En el desarrollo de mecanismos de interacción para la realidad virtual, las principales consideraciones son:

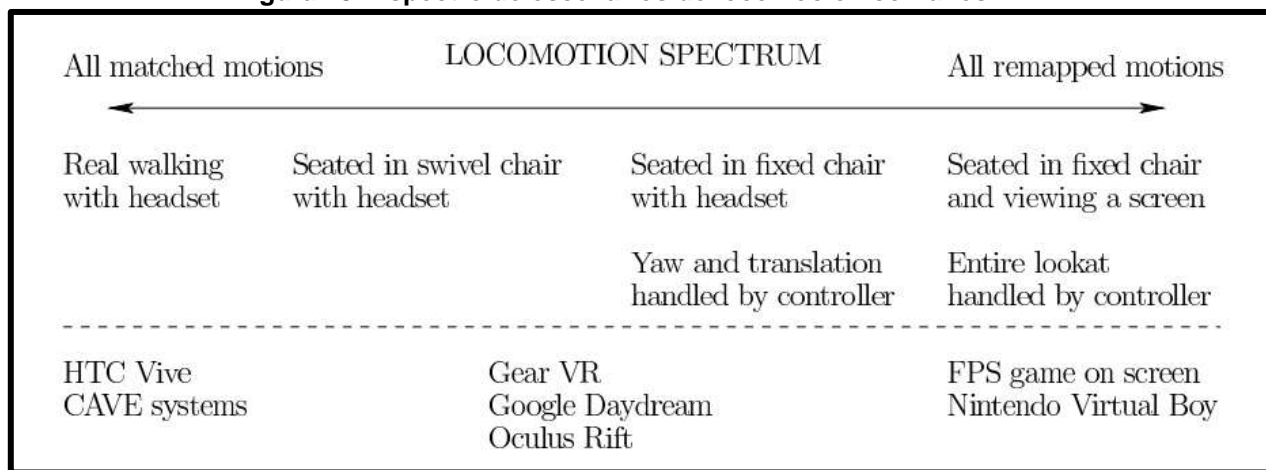
1. Efectividad para la tarea en términos de lograr la velocidad, precisión y rango de movimiento requeridos, si corresponde.
2. Dificultad para aprender los nuevos programas motores; idealmente, no se debe esperar que el usuario pase muchos meses dominando un nuevo mecanismo.
3. Facilidad de uso en términos de carga cognitiva; en otras palabras, el mecanismo de interacción debería requerir poca o ninguna atención enfocada después de alguna práctica.

4. Comodidad general durante el uso durante períodos prolongados; el usuario no debe desarrollar fatiga muscular, a menos que la tarea sea hacer algo de ejercicio físico.

2.2.3.1. Locomoción.

Suponga que el mundo virtual cubre un área mucho más grande que la parte del mundo real que se rastrea. En otras palabras, la zona coincidente es pequeña en relación con el mundo virtual. En este caso, se necesita algún tipo de mecanismo de interacción para mover al usuario en el mundo virtual mientras ella permanece fija dentro del área rastreada en el mundo real. Un mecanismo de interacción que mueve al usuario de esta manera se llama locomoción. Es como si el usuario viajara en un vehículo virtual que se dirige a través del mundo virtual.

Figura 13: Espectro de escenarios de locomoción comunes



Fuente: LaValle, 2019

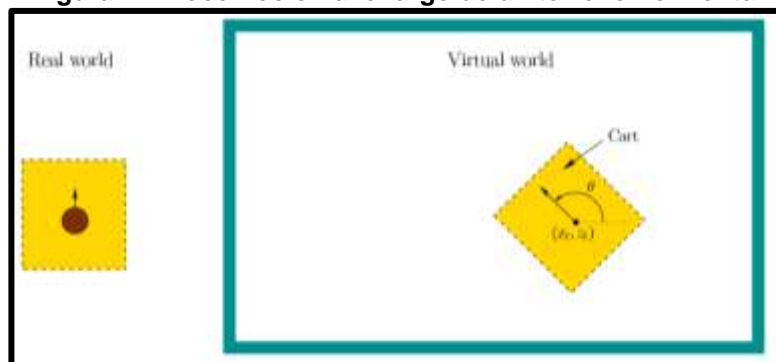
Como se puede observar en la Figura 13 existe un espectro de escenarios de locomoción comunes. A la izquierda, el usuario camina en un espacio abierto mientras usa auriculares. No se necesita locomoción a menos que el mundo virtual sea más grande que el espacio abierto. Este caso no implica desajuste entre movimientos reales y virtuales. Los dos casos centrales corresponden a un usuario sentado que usa auriculares. En estos casos, se utiliza un mecanismo de interacción para cambiar la posición de la zona coincidente en el mundo virtual. Si el usuario está sentado en una silla giratoria, entonces podría cambiar la dirección a la que se enfrenta (orientación de guiñada) girando la silla. Esto puede considerarse como

orientar el torso del usuario en el mundo virtual. Si el usuario está sentado en una silla fija, entonces la orientación del torso virtual generalmente cambia usando un controlador, lo que resulta en una mayor falta de coincidencia. (LaValle, 2019, pp: 287-293)

2.2.3.2. Caminar redirigido.

Si se rastrea al usuario a través de un espacio muy grande, como una región cuadrada de al menos 30 metros a cada lado, entonces es posible hacerle pensar que está caminando en línea recta durante kilómetros mientras en realidad está caminando círculos. Esta técnica se llama caminar redirigido. Caminar a lo largo de una línea recta a través de largas distancias sin señales visuales es prácticamente imposible para los humanos (¡y para los robots!) Porque en el mundo real es imposible lograr una simetría perfecta. Una dirección tenderá a dominar a través de un desequilibrio en la fuerza motora y las señales sensoriales, haciendo que las personas viajen en círculos. (LaValle, 2019, pp: 294-299)

Figura 14: Locomoción a lo largo de un terreno horizontal



Fuente: LaValle, 2019

2.2.3.3. Locomoción no plana.

Ahora considere casos de locomoción más complicados. Si el usuario camina sobre un terreno, entonces el componente y se puede aumentar o disminuir simplemente para reflejar el cambio de altitud. Esto puede parecer realista, pero tenga en cuenta que aumenta la cantidad de desajustes entre los mundos real y virtual porque la vección vertical se combina con la vección hacia adelante. En el caso de moverse a través de un medio 3D, Las configuraciones comunes incluyen una nave espacial virtual, un avión o un buzo. La vección de guiñada, cabeceo y balanceo se puede

generar fácilmente. Por ejemplo, imagine volar una nave espacial virtual. Al hacer rodar la embarcación, se puede producir la vección del rollo cuando las estrellas giran en un patrón circular. Si un desarrollador debe hacer un movimiento artesanal de esta manera, entonces se deben seguir las sugerencias anteriores para reducir la intensidad de la vección. Además, se debe realizar una experimentación cuidadosa con sujetos humanos para determinar qué formas de vección son peores en la aplicación particular; Para evitar singularidades, para los sistemas en los que son posibles las 3 DOF de movimiento rotacional, las transformaciones virtuales del vehículo se mantienen mejor en términos de cuaterniones. Agregar efectos especiales que muevan el punto de vista causará más dificultades con la vección. Por ejemplo, hacer que un avatar salte hacia arriba y hacia abajo provocará una vección vertical. También es una mala idea tener en cuenta los movimientos de la cabeza que se balancean al caminar debido al aumento de la falta de coincidencia. Imagine un caso mucho peor de mirar a través de los ojos de un avatar que realiza gimnasia. La visión del mundo puede volverse insoportable durante múltiples vueltas. (LaValle, 2019, pp: 299)

2.2.3.4. Hardware especializado.

Se han desarrollado muchos tipos de hardware para soportar la locomoción. Uno de los ejemplos más antiguos es crear una cabina completa para la simulación de vuelo de una aeronave.

Figura 15: (a) Una cinta de correr omnidireccional utilizada en un sistema CAVE (cueva) por el ejército de los EE. UU. para entrenamiento. (b) Un sistema para montar bicicleta en casa conectado a un auricular VR.



Fuente: LaValle, 2019

2.2.3.5. Teletransportación.

Los métodos de locomoción cubiertos hasta ahora se han centrado principalmente en reproducir experiencias que son familiares en el mundo real, que proporcionan ejemplos del principio de simulación universal. En realidad, virtual, sin embargo, también podríamos movernos de formas que no son plausibles físicamente. El más común es la teletransportación, que funciona como un transportador en la serie de televisión StarTrek. El usuario es transportado inmediatamente a otra ubicación. Para seleccionar una ubicación donde el usuario preferiría pararse, simplemente podría apuntar el láser virtual y presionar una tecla para ser teletransportado instantáneamente. Para facilitar el apuntar al piso, el rayo podría ser un arco parabólico que sigue a la gravedad, similar a una corriente de agua. El mecanismo de teletransportación reduce la vección y, por lo tanto, la enfermedad de VR; sin embargo, puede tener el costo de un aprendizaje reducido de la disposición espacial del entorno. Al realizar la teletransportación, es importante no cambiar la orientación de guiñada del punto de vista; de lo contrario, el usuario puede volverse más desorientado. Es posible que no entienda dónde está ahora posicionado y orientado en el mundo virtual en relación con la ubicación anterior. Tenga en cuenta que el principio de simulación universal puede emplearse una vez más para tomar prestadas herramientas de navegación efectivas del mundo real. Si los edificios virtuales y las ciudades se presentan de manera común en el mundo real, entonces deberían ser más fáciles de navegar. Incluso se pueden colocar letreros y puntos de referencia en el mundo virtual para ayudar con la navegación. En el mundo real, los letreros a menudo nos dicen la ubicación de las salidas, los nombres de las calles o los límites de un distrito. Los puntos de referencia como edificios altos, molinos de viento o torres proporcionan señales visuales que son efectivas para navegar. (LaValle, 2019, pp: 300)

Figura 16: Teletransportación



Fuente: LaValle, 2019

2.2.3.6. Wayfinding.

El problema cognitivo de aprender una representación espacial y usarla para navegar se llama orientación. Este es un proceso de nivel superior que el mecanismo de locomoción de bajo nivel, pero los dos están estrechamente relacionados. Un problema con los sistemas de locomoción que no son familiares en el mundo real es que los usuarios podrían no aprender la disposición espacial del mundo que los rodea. ¿Seguiría tu cerebro formando células de lugar para un entorno en el mundo real si pudieras teletransportarte de un lugar a otro? Observamos ampliamente este fenómeno con personas que aprenden a navegar por una ciudad utilizando solo GPS o servicios de taxi, en lugar de hacer su propio camino. (LaValle, 2019, pp: 300)

2.2.3.7. Manipulación.

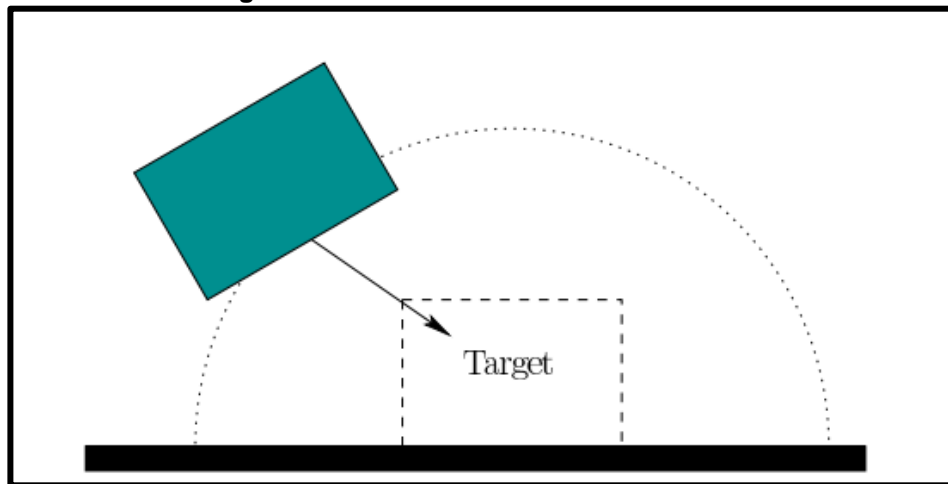
Interactuamos con objetos en el mundo real por muchas razones. Puede comer un tazón de sopa moviendo una cuchara entre el tazón y la boca. Puede recoger una piedra y lanzarla lo más lejos posible. Puede ponerse un par de pantalones. Estos ejemplos y muchos más caen bajo el tema de la manipulación. En el mundo real, la manipulación implica relaciones sensoriomotoras complejas que, a través de la evolución y la experiencia, nos permiten manipular objetos en una amplia variedad de entornos. La variación de los objetos incluye diferencias en tamaño, peso, fricción, flexibilidad, temperatura, fragilidad, etc. De alguna manera nuestros cuerpos pueden

manejar eso. Hacer que los robots realicen la manipulación de la forma en que lo hacen los humanos ha sido un camino largo y frustrante, con un éxito limitado. Debido a la complejidad de la manipulación en el mundo real, es un candidato ideal para aplicar los conceptos de reasignación para hacer que la manipulación sea lo más simple posible en realidad virtual. El mundo virtual no tiene que seguir la complicada física de la manipulación. En cambio, es preferible realizar operaciones como seleccionar, agarrar, manipular, transportar y colocar un objeto lo más rápido y fácil posible. Además, se debe evitar el alcance extenso u otras formas de tensión muscular, a menos que la experiencia de RV esté diseñada para proporcionar ejercicio. (LaValle, 2019, pp: 302)

2.2.3.8. Selección.

Una de las formas más simples de seleccionar un objeto en el mundo virtual es con el puntero láser virtual. Varias variaciones pueden ayudar a mejorar el proceso de selección. Por ejemplo, el usuario podría en su lugar sostener una linterna virtual que ilumina posibles selecciones. El campo de visión de la linterna podría ser ajustable. Se puede colocar un espejo virtual para que se pueda hacer una selección en una esquina. Con un puntero, el usuario simplemente ilumina el objeto de interés y presiona un botón. Si el objetivo es recuperar el objeto, se puede colocar inmediatamente en la mano virtual o en el inventario del usuario. Si el objetivo es manipular el objeto de manera repetitiva estándar, entonces presionar el botón podría hacer que se ejecute un programa motor virtual. Esto podría usarse, por ejemplo, para girar el pomo de una puerta, abriendo así una puerta. En usos como este, los desarrolladores pueden querer establecer un límite en la profundidad del puntero láser, de modo que el usuario debe estar lo suficientemente cerca como para permitir la interacción. ¡Puede parecer inapropiado, por ejemplo, girar los pomos de las puertas desde el otro lado de la habitación! Si el objeto es difícil de ver, entonces el proceso de selección puede ser complicado. Puede estar detrás de la cabeza del usuario, lo que puede requerir un giro incómodo.

Figura 17: Definir una cuenca de atracción



Fuente: LaValle, 2019

Si el usuario lleva un objeto a una gran distancia, no es necesario que apriete o agarre el controlador; Esto produciría fatiga innecesaria. En algunos casos, se puede esperar que el usuario inspeccione cuidadosamente el objeto mientras lo posee. Por ejemplo, es posible que desee moverlo en su mano para determinar su estructura 3D. La orientación del objeto podría configurarse para seguir exactamente la orientación 3D de un controlador que tiene el usuario. El usuario incluso podría sostener un objeto real en la mano que es rastreado por cámaras externas, pero tiene una apariencia diferente en el mundo virtual. Tenga en cuenta que un objeto podría incluso manipularse directamente en su lugar original en el mundo virtual, sin acercarlo al cuerpo virtual del usuario. En este caso, la mano virtual se lleva al objeto, mientras que la mano física permanece en su lugar. Tener un brazo más largo de lo normal también se puede simular para recuperar y colocar objetos a mayores distancias. (LaValle, 2019, pp: 302)

2.2.3.9. Colocación.

Ahora considere desabrochar el objeto y colocarlo en el mundo. Un caso fácil para el usuario es presionar un botón y hacer que el objeto simplemente caiga en el lugar correcto. Esto se logra mediante una cuenca de atracción que es una función potencial atractiva definida en una vecindad de la pose objetivo (posición y orientación); ver Figura 17. El mínimo de la función potencial está en el objetivo. Después de soltar el objeto, el objeto cae en la pose objetivo moviéndose de modo

que el potencial se reduzca al mínimo. Este comportamiento se ve en muchos programas de dibujo 2D para que los puntos finales de los segmentos de línea se encuentren convenientemente. Un ejemplo de colocación conveniente de objetos es el juego de sandbox Minecraft 2011 de Markus Persson (Notch), en el que los bloques de construcción simplemente encajan. Los niños han construido millones de mundos virtuales de esta manera. (LaValle, 2019, pp: 304)

Figura 18: (a) Mandos oculus rift. (b) Utilización del Oculus Rift



Fuente: LaValle, 2019

2.2.3.10. Reasignación.

Ahora considere el poder de la reasignación, El caso más simple es el uso del botón para seleccionar, agarrar y colocar objetos. En lugar de un botón, el usuario podría generar movimientos continuos y rastrearlos por los sistemas. Los ejemplos incluyen girar una perilla, mover una barra deslizante, mover un dedo sobre una pantalla táctil y mover un cuerpo flotante por el espacio. Recuerde que uno de los aspectos más importantes de la reasignación es la fácil capacidad de aprendizaje. Reducir el número de grados de libertad que se reasignan generalmente facilitará el proceso de aprendizaje. Para evitar los brazos de gorila y los problemas relacionados, se podría imponer un factor de escala en el dispositivo rastreado para que una pequeña cantidad de cambio de posición en el controlador corresponda a un gran movimiento en el mundo virtual. Este problema podría estudiarse nuevamente utilizando la ley de Fitts como en el caso del mouse de la computadora. Tenga en cuenta que esto podría tener un efecto adverso sobre la precisión en el mundo virtual. En algunos ajustes, la escala de orientación también puede ser deseable. En este caso, la

velocidad angular 3D (ω_x , ω_y , ω_z) podría ser escalada por un factor para inducir más rotación en el mundo virtual que en el mundo real. (LaValle, 2019, pp: 305)

2.2.4. Evaluación de sistemas de realidad virtual y experiencias.

¿Qué auricular es mejor? ¿Qué experiencia de realidad virtual es más cómoda durante un largo período de tiempo? ¿Cuánto campo de visión es suficiente? ¿Cuál es el mecanismo de interacción más apropiado? Los ingenieros y desarrolladores quieren saber las respuestas a este tipo de preguntas; sin embargo, debe quedar claro en este punto que son difíciles de responder debido a la forma en que la fisiología y la percepción humanas operan e interactúan con los sistemas de ingeniería. Por el contrario, preguntas de ingeniería pura, como "¿Cuál es la duración estimada de la batería?" o "¿Cuál es la velocidad máxima del vehículo en terreno llano?", son mucho más accesibles.

Cuando los científicos aplican la realidad virtual para estudiar las estructuras neurales y la percepción de una rata, existe una clara separación entre la rata y el científico. Sin embargo, en el caso de la realidad virtual para humanos, el desarrollador frecuentemente prueba sus propias creaciones. En este caso, el desarrollador alterna entre el papel de científico y rata. Esto introduce numerosos problemas, especialmente si el desarrollador es ingenuo sobre los problemas de percepción. Para complicar aún más las cosas es la adaptación, que ocurre en todas las escalas. Por ejemplo, una persona que evalúa una experiencia de realidad virtual muchas veces durante varias semanas inicialmente puede encontrarla incómoda, pero luego se acostumbra a ella. Por supuesto, esto no implica que su probabilidad de enfermar a un nuevo usuario sea menor. También hay una gran variación entre las personas. Cualquier persona, incluido el desarrollador, proporciona un solo punto de datos. Las personas que son inmunes a la enfermedad por vección no tendrán problemas para desarrollar tales sistemas e infligirlos a otros. (LaValle, 2019, pp: 341)

2.2.4.1. Entrenamiento perceptivo.

La mayoría de las personas que prueban la realidad virtual por primera vez desconocen las fallas técnicas que serían obvias para algunos ingenieros y

desarrolladores experimentados. Si la experiencia de realidad virtual funciona como debería, entonces el usuario debe sentirse abrumado por los estímulos visuales dominantes y sentir que está habitando el mundo virtual.

Es posible que algunas partes no funcionen como se diseñaron o que se hayan descuidado algunos problemas de percepción. Esto podría resultar en una experiencia que no es tan buena como podría haber sido después de realizar algunos ajustes simples. Peor aún, las fallas pueden causar que el usuario se sienta fatigado o enfermo. Al final, estos usuarios generalmente no son conscientes de lo que salió mal. Podrían culpar a cualquier cosa, como estímulos visuales particulares, una experiencia particular, el hardware de los auriculares o incluso todo el concepto de realidad virtual.

Este problema puede mitigarse capacitando a usuarios y desarrolladores específicos para que noten los tipos comunes de fallas. Al desarrollar un programa de capacitación perceptiva, puede solicitar un usuario que busque un artefacto o defecto en particular, o que practique repetidamente realizar alguna tarea. Por ejemplo, si se presenta un estímulo constante durante un largo período de tiempo, entonces su intensidad percibida crítica.

A través de la exposición repetida y guiada a un sistema y experiencia de realidad virtual en particular, los usuarios pueden adaptar sus sistemas de percepción. Esta es una forma de aprendizaje perceptual, que es una rama de la psicología perceptiva que estudia los cambios duraderos en los sistemas perceptivos de un organismo en respuesta a su entorno. A medida que la realidad virtual se convierte en un nuevo entorno para el organismo, las oportunidades y los límites del aprendizaje perceptivo permanecen en gran parte inexplorados. A través de activo entrenamiento, la forma en que los usuarios se adaptan se puede controlar para que aumenten sus capacidades de percepción y poder de discriminación. Esto a su vez se puede utilizar para evaluar a los capacitadores que brindan comentarios frecuentes en el proceso de desarrollo. Una alternativa es desarrollar un sistema automatizado que pueda detectar fallas sin intervención humana. Es probable que una combinación de

evaluación humana y automática sea importante en los próximos años. (LaValle, 2019, pp: 341-342)

2.2.5. Generación de terreno procesal.

La generación de terreno procesal es un campo ampliamente explorado debido a su importancia en la industria del entretenimiento, más específicamente en películas y videojuegos. Es imperativo que tales aplicaciones generen paisajes visualmente atractivos y ricos en detalles. Si bien los paisajes físicamente plausibles pueden generarse mediante la erosión hidráulica y eólica, y las simulaciones tectónicas de elevación, tales enfoques no son factibles para generar datos de elevación en tiempo real y, a menudo, carecen de capacidad de control. Por otro lado, el terreno puede ser bosquejado manualmente o completamente cargado por datos de entrada. Sin embargo, estos enfoques no son escalables para terrenos extremadamente grandes, que pueden extenderse varios kilómetros a la resolución de centímetros.

2.2.5.1. Método Pink Noise.

Una técnica utilizada a menudo para la generación rápida del terreno es simular $1/f$ de ruido (también conocido como "Pink Noise ") que se caracteriza porque la densidad de energía espectral es proporcional a la recíproca de la frecuencia, es decir:

$$P(f) = \frac{1}{f^a}$$

Ec - 1

Donde:

$P(f)$: es la función de potencia de la frecuencia.

f : es la frecuencia.

α : es una variable cerca de 1.

Este tipo de ruido se aproxima al terreno montañoso sin protección del mundo real bien y se ha utilizado ampliamente en gráficos por computadora durante las últimas décadas. A continuación, se analizan dos métodos para generar ruido de tipo $1/f$, síntesis espectral y desplazamiento del punto medio. Al generar una base de terreno para que trabajen los algoritmos de erosión, vale la pena señalar que el más cerca de la base del terreno al resultado deseado, cuanto menos trabajo requiere el (a menudo cálculo pesado) algoritmo de erosión en sí mismo. Para ayudar a crear una base del terreno con mejores características de erosionadas terreno, el uso de diagramas de Voronoi y el filtrado de perturbaciones. (Jacob Olsen, 2004, pp: 2).

2.2.5.2. Diagramas de voronoi.

El problema con el uso de ruido $1/f$ para simular el terreno del mundo real es que es estadísticamente homogéneo e isotrópico, propiedades que el terreno real No comparte. Una forma de romper la monotonía. y controlar las principales características del paisaje son los diagramas de Voronoi cuyo uso en la generación de texturas de procedimiento ha sido descrito por Steven Worley. Se pueden usar diagramas de Voronoi para una variedad de efectos al crear procedimientos texturas: la mayoría de las variantes se asemejan a algún tipo de estructuras similares a células que se pueden usar para simular tejido, esponja, escamas, guijarros, losas, o en este caso, montañas enteras.

La implementación funciona por dividiendo el mapa en regiones y luego al azar coloque una cantidad de "puntos de características" en cada región. Para cada celda en el mapa, un conjunto de valores d_n , $n = 1, 2, 3, \dots$ se calculan de acuerdo con una métrica de distancia definida para que d_1 sea la distancia al punto de característica más cercano, d_2 es la distancia a el siguiente punto de distancia más cercano, etc. Combinaciones lineales de la forma:

$$h = c_1d_1 + c_2d_2 + c_3d_3 + \cdots + c_nd_n$$

Ec - 2

Donde:

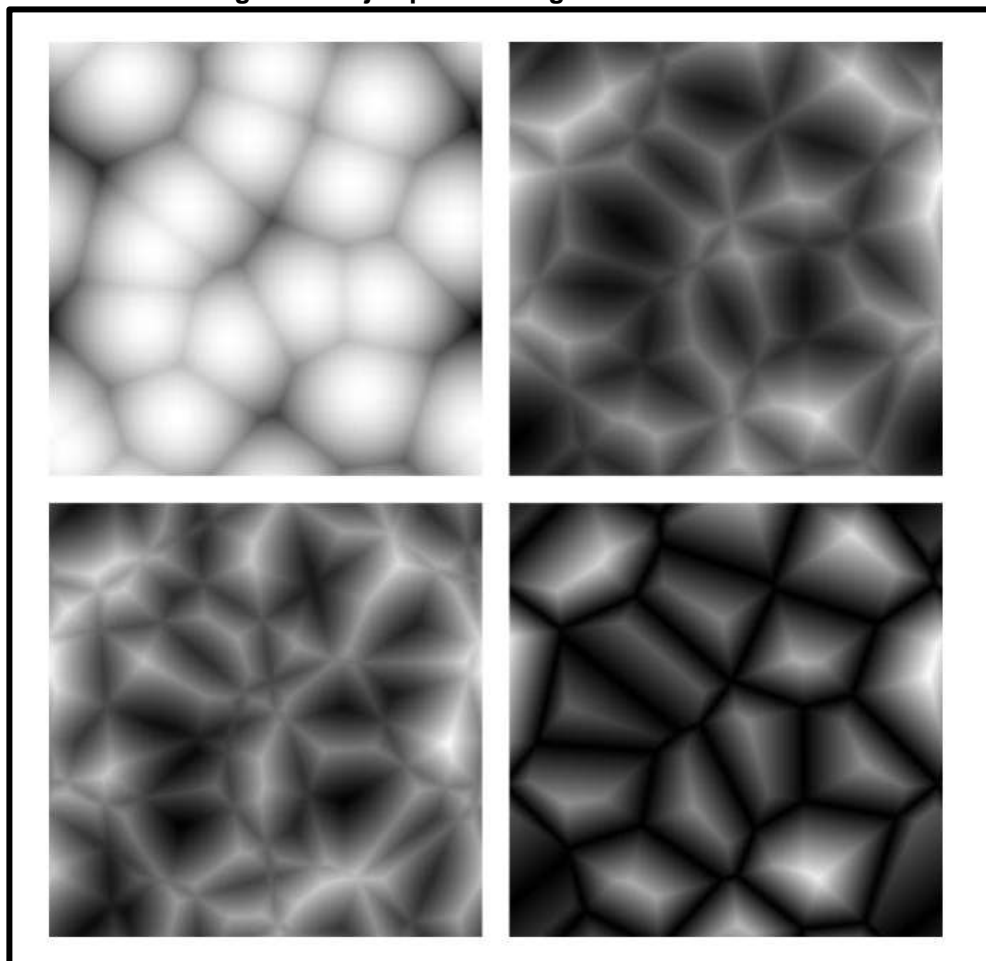
h : Altura de celda en el mapa.

c_n : Coeficientes característicos.

d_n : Conjunto de valores de distancia.

con coeficientes c_1, \dots, c_n entonces producirá las estructuras celulares; Para crear características montañosas, los coeficientes $c_1 = -1$ y $c_2 = 1$ (con el resto son ceros) se utilizan, ya que puede agregar distintas líneas de crestas y cauces conectados a el terreno. Estos valores también dan el Voronoi diagramas otra propiedad útil que será

Figura 19: Ejemplos de diagramas de Voronoi



Fuente: LaValle, 2019

coeficientes $c1 = -1$ (arriba a la izquierda), $c2 = 1$ (arriba derecha), $c3 = 1$ (abajo a la izquierda), $c1 = -1$ y $c2 = 1$ (abajo a la derecha).

Normalmente, las distancias están determinadas por la métrica de distancia euclidiana.

$$d = \sqrt{dx^2 + dy^2}$$

Ec - 3

que es bastante lento debido a la raíz cuadrada. Cambiar la métrica de distancia a

$$d = dx^2 + dy^2$$

Ec - 4

Donde:

d : Distancia

dx : Distancia en x

dy : Distancia en y

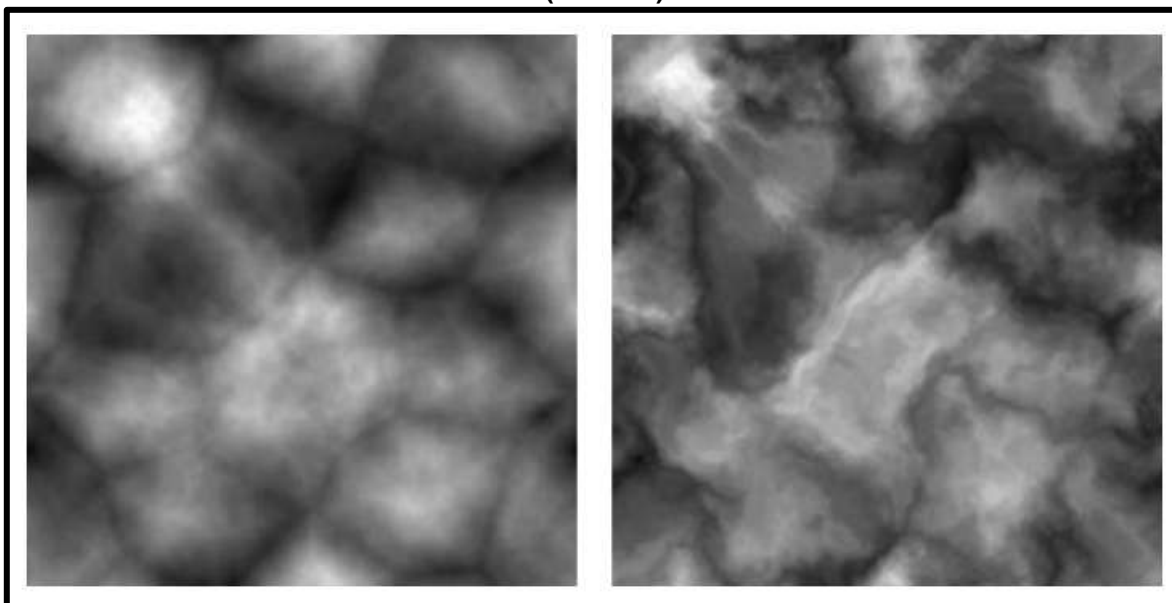
Produce una gran aceleración, la diferencia en el mapa de altura resultante es insignificante. Esta optimización junto con una reducción en el radio de búsqueda al encontrar la función más cercana puntos (que ocasionalmente produce errores menores) reduce el tiempo de cálculo a un tercio.

2.2.5.3. Combinación y perturbación.

Aunque los diagramas de Voronoi tienen algunas útiles propiedades de las que carece el ruido $1/f$, no sustituyen las funciones de ruido. Los mejores resultados se logran con alguna combinación de ambos; en este caso dos tercios alisó el cuadrado de diamante ruido de método y un tercer diagrama de Voronoi con Se utilizarán los coeficientes $c1 = -1$ y $c2 = 1$. Esta combinación se conoce como la altura combinada mapa. Para arrugar las líneas rectas del diagrama de Voronoi, un filtro de perturbación. Este filtro funciona utilizando una función de ruido (similar a las descritas arriba) para calcular un desplazamiento con azar distancia y dirección para

cada celda. El mapa de altura combinado antes y después de la perturbación se puede ver en la Figura 9. La magnitud de la el filtrado de perturbaciones se establece en 0.25, lo que significa que un punto dado en el mapa de altura no se puede desplazar más de $N4/4$ células. La perturbación que se filtra también aumenta el puntaje de erosión porque algunas áreas están estiradas y algunos están comprimidos, lo que aumenta os.

Figura 20: El mapa de altura combinado antes de la perturbación (izquierda) y después (derecha).



Fuente: LaValle, 2019

2.3. UNITY.

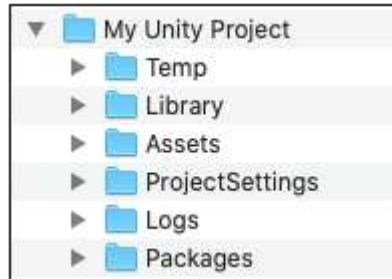
Unity es la plataforma de desarrollo 3D en tiempo real que permite que artistas, diseñadores y desarrolladores trabajen juntos para crear experiencias interactivas y envolventes increíbles.

2.3.1. Importado.

Para incorporar los Activos creados fuera de Unity a su Proyecto Unity exportando el archivo directamente a la carpeta Activos debajo de su Proyecto, o copiándolo en esa carpeta. Para muchos formatos comunes, se puede guardar su archivo fuente directamente en la carpeta de Activos de su Proyecto y Unity puede leerlo. Unity también detecta cuándo guarda nuevos cambios en el archivo y vuelve a importar los archivos según sea necesario.

Cuando se crea un proyecto de Unity, se está creando una carpeta (llamada así por su proyecto) que contiene las siguientes subcarpetas:

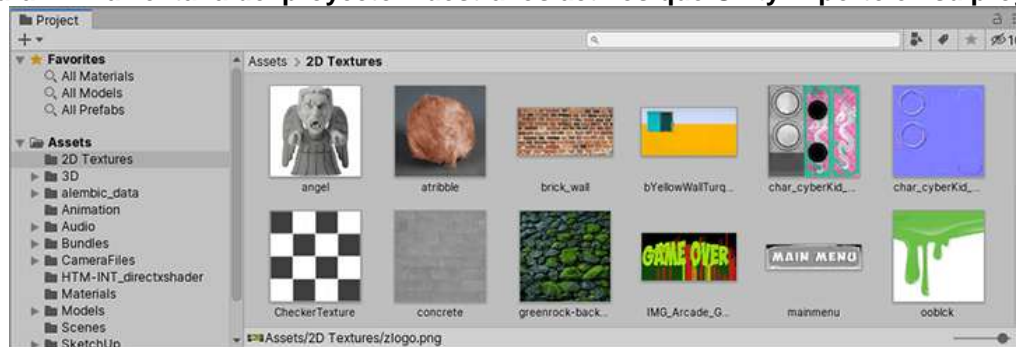
Figura 21: La estructura básica de archivos de un proyecto de Unity



Fuente: Unity, 2020

Unity detecta automáticamente los archivos a medida que se agregan a la carpeta Activos o si se modifican. Cuando coloca cualquier activo en su carpeta de activos, aparece en su vista de proyecto.

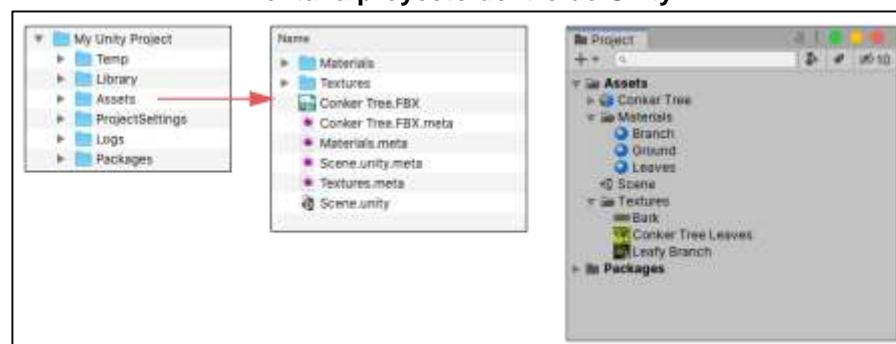
Figura 22: La ventana del proyecto muestra los activos que Unity importó en su proyecto



Fuente: Unity, 2020

Si arrastra un archivo a la ventana Proyecto de Unity desde su computadora (ya sea desde el Finder en Mac o desde el Explorador en Windows), Unity copiará en su carpeta de Activos y aparece en la ventana Proyecto. (Unity, 2020)

Figura 23: La relación entre la carpeta de activos en su proyecto Unity en su computadora y la ventana proyecto dentro de Unity



Fuente: Unity, 2020

Los elementos que aparecen en la ventana del Proyecto representan (en la mayoría de los casos) archivos reales en su computadora, y si los elimina dentro de Unity, también los está eliminando de su computadora.

La forma más sencilla de mover o cambiar el nombre de sus activos de forma segura es hacerlo siempre desde la carpeta Proyecto de Unity. De esta manera, Unity mueve o cambia automáticamente el nombre del meta archivo correspondiente. Si lo desea, puede leer más sobre los archivos. meta y lo que sucede detrás de escena durante el proceso de importación.

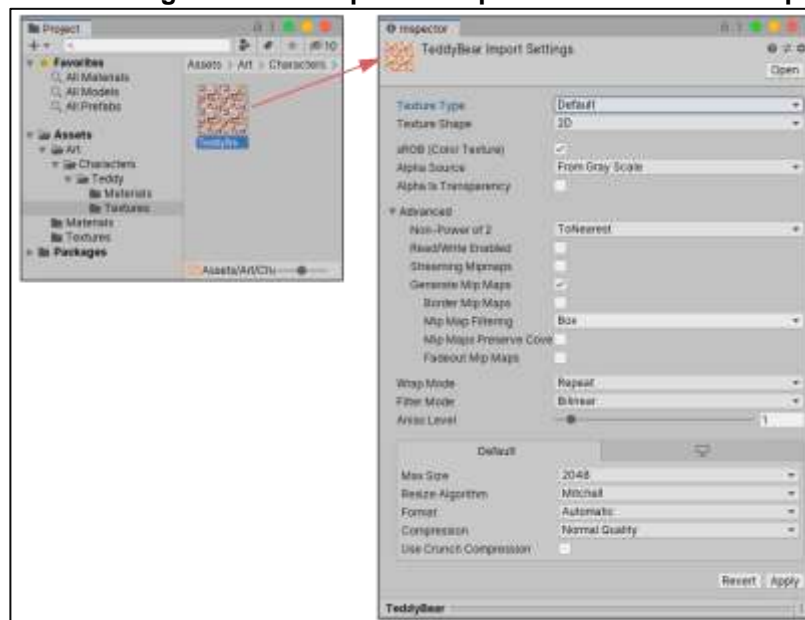
Para incorporar colecciones de activos a su proyecto, puede usar paquetes de activos. (Unity, 2020)

2.3.1.1. Inspección de activos.

Cada tipo de activo que admite Unity tiene un conjunto de configuraciones de importación, que afectan la apariencia o el comportamiento del activo. Para ver la configuración de importación de un activo, seleccione el activo en la vista de proyecto. La configuración de importación para este activo aparecerá en el Inspector

Las opciones que aparecen varían según el tipo de activo seleccionado.

Figura 24: Configuración de importación para ese activo en el inspector

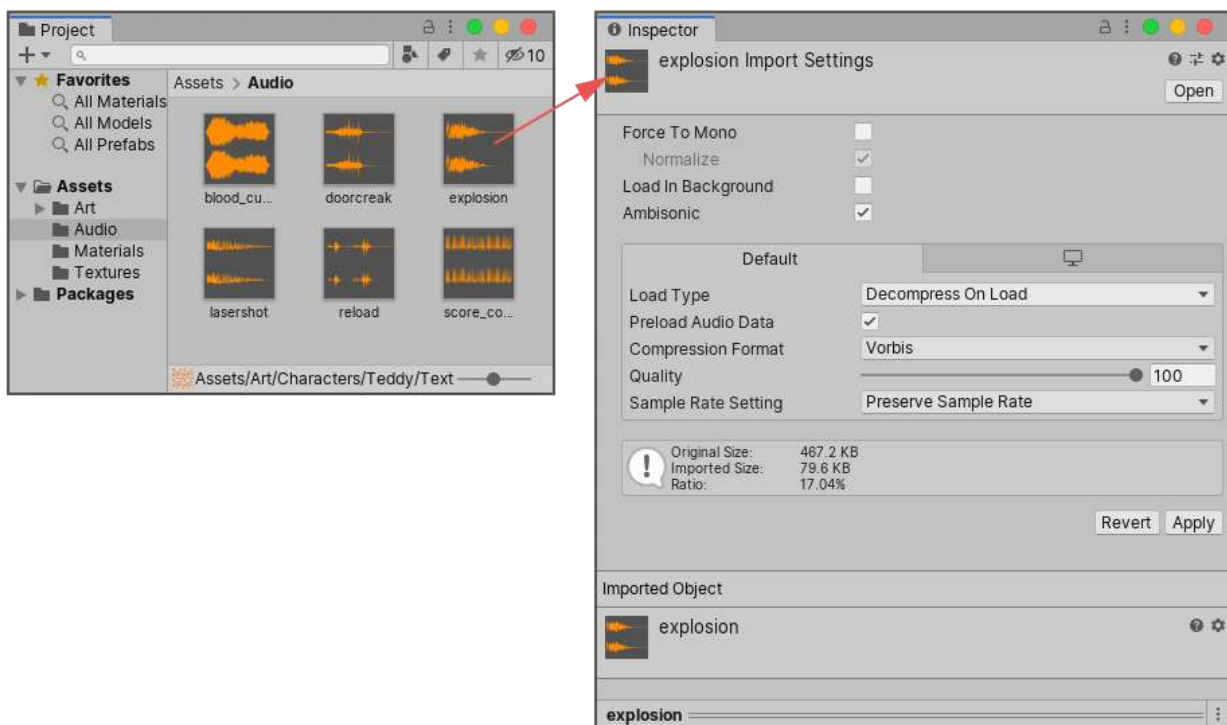


Fuente: Unity, 2020

La configuración de importación para un archivo FBX permite ajustar la escala, generar normales o mapas de luz, coordenadas y clips de animación de división y recorte definido en el archivo.

Para otros tipos de activos, la configuración de importación se ve diferente. Las diversas configuraciones que se relacionan con el tipo de activo seleccionado. Aquí hay un ejemplo de un activo de audio, con su configuración de importación relacionada que se muestra en el inspector:

Figura 25: Un activo de audio seleccionado en la ventana del proyecto



Fuente: Unity, 2020

Un activo de audio seleccionado en la ventana del proyecto muestra la configuración de importación de audio para ese activo en el Inspector.

Si está desarrollando un proyecto multiplataforma, se podrá anular la configuración "predeterminada" y asignar diferentes configuraciones de importación por plataforma. (Unity, 2020)

2.3.1.2. Importado de modelos.

Los archivos de modelo pueden contener una variedad de datos, como mallas de personajes y terrenos, plataformas de animación y clips, así como materiales y texturas. Lo más probable es que el archivo no contenga todos estos elementos a la vez, pero puede seguir cualquier parte del flujo de trabajo que necesite:

- Seleccionar el archivo en la vista Proyecto para ver la ventana Configuración de importación.
- Establecer cualquier opción de importador general o específica del modelo.
- Configurar las opciones para importar plataformas y animaciones (no disponible para los modelos SpeedTree).
- Manejar los materiales y texturas.
- Arrastra el archivo a Unity.

2.3.1.3. Configuración de opciones de importador generales y específicas.

Las opciones disponibles para los modelos SpeedTree frente a otros modelos son muy diferentes. Por ejemplo, la pestaña Modelo SpeedTree proporciona opciones principalmente para configurar transiciones entre los niveles. Los personajes y los modelos animados ofrecen opciones más diversas en su pestaña Modelo, que le permiten:

- Use las propiedades Factor de escala y Convertir unidades para ajustar cómo Unity interpreta las unidades. Por ejemplo, 3ds Max usa 1 unidad para representar 10 centímetros, mientras que Unity usa 1 unidad para representar 1 metro.
- Use las propiedades Compresión de malla, Lectura / escritura habilitada, Optimizar malla, conservar cuadrículas, Formato de índice o Vértices de soldadura para reducir recursos y ahorrar memoria.
- Se puede habilitar la opción Importar BlendShapes si el archivo Modelo vino de Maya o 3ds Max, o cualquier otra aplicación de modelado 3D que admita la animación de metamorfosis.

- Se puede habilitar la opción Generar colisionadores para la geometría del entorno.
- Se puede habilitar configuraciones específicas de FBX, como Importar visibilidad o Importar cámaras e Importar luces.
- Para los archivos de Modelo que contienen solo Animación, se puede habilitar la opción Conservar jerarquía para evitar que la jerarquía no coincida en su esqueleto.
- Se puede configurar Intercambiar UV y Generar UV de Lightmap si está utilizando un Lightmap.
- Puede ejercer control sobre cómo Unity maneja las normales y las tangentes en su modelo con las normales, Modo Normal, Tangentes o Ángulo de suavizado.

2.3.1.4. Configuración de opciones para importar plataformas y animaciones.

Si el archivo contiene datos de Animación, puede seguir las pautas para configurar el Aparejo usando la pestaña Aparejo y luego extraer o definir Clips de Animación usando la pestaña Animación. El flujo de trabajo difiere entre los tipos de animación Humanoide y Genérico (no Humanoide) porque Unity necesita que la estructura ósea del humanoide sea muy específica, pero solo necesita saber qué hueso es el nodo raíz para el tipo Genérico:

- Flujo de trabajo de tipo humanoide
- Flujo de trabajo de tipo genérico

2.3.2. Manejo de materiales y texturas.

Si su archivo contiene Material o Textura, puede definir cómo desea tratar con ellos:

- Haga clic en la pestaña Materiales en la ventana Configuración de importación.
- En el menú desplegable Modo de creación de material, elija cómo desea importar los Materiales desde el archivo FBX. A menos que elija Ninguno, aparecen varias opciones en la pestaña Materiales, incluida la opción Ubicación, cuyo valor determina cuáles son las otras opciones.

- Elija la opción usar materiales incrustados para mantener los materiales dentro del activo importado.
- Cuando haya terminado de configurar las opciones, haga clic en el botón Aplicar en la parte inferior de la ventana configuración de importación para guardarlas o haga clic en el botón Revertir para cancelar.

2.3.3. Entradas.

La entrada le permite al usuario controlar su aplicación usando un dispositivo, toque o gestos. Puede programar elementos integrados en la aplicación, como la interfaz gráfica de usuario (GUI) o un avatar de usuario, para responder a la entrada del usuario de diferentes maneras.

Unity admite la entrada de muchos tipos de dispositivos de entrada, incluidos:

- Teclados y ratones
- Joysticks
- Controladores
- Pantallas táctiles
- Capacidades de detección de movimiento de dispositivos móviles, como acelerómetros o giroscopios
- VR y AR
- Controladores

Unity admite la entrada a través de dos sistemas separados:

- El administrador de entrada forma parte de la plataforma central de Unity y está disponible de forma predeterminada.
- El Sistema de entrada es un paquete que debe instalarse a través del Administrador de paquetes antes de poder usarlo. Requiere el tiempo de ejecución de .NET 4, y no funciona en proyectos que usan el tiempo de ejecución de .NET 3.5 anterior.

2.3.3.1. Unidad de entrada XR.

Unity proporciona información sobre todos los dispositivos de entrada compatibles con Unity para realidad virtual, realidad aumentada y aplicaciones de realidad mixta de Windows. Esta página cubre los siguientes temas:

- Asignaciones de entrada XR
- Acceso a dispositivos de entrada
- Acceso a funciones de entrada en un dispositivo de entrada
- Acceso a la entrada XR a través del sistema de entrada heredado
- Hápticos

Las plataformas XR tienen una gran variedad de características de entrada que puede aprovechar cuando diseña interacciones de usuario. Su aplicación puede usar datos específicos que hacen referencia a posiciones, rotaciones, tacto, botones, joysticks y sensores de dedo. Sin embargo, el acceso a estas funciones de entrada puede variar mucho entre plataformas. Por ejemplo, existen pequeñas diferencias entre Vive y Oculus Rift, pero una plataforma de escritorio habilitada para VR y una plataforma móvil como Daydream difieren mucho más.

Unity proporciona una estructura C # llamada `InputFeatureUsage`, que define un conjunto estándar de controles de dispositivos físicos (como botones y disparadores) para acceder a la entrada del usuario en cualquier plataforma. Estos le ayudan a identificar los tipos de entrada por nombre.

Cada uno `InputFeatureUsage` corresponde a una acción o tipo de entrada común. Por ejemplo, Unity define el `InputFeatureUsage` llamado `trigger` como una entrada de un solo eje que controla el dedo índice, independientemente de qué XR plataforma que utiliza. Puede usar `InputFeatureUsage` para obtener el `trigger` estado usando su nombre, por lo que no necesita configurar un eje (o un botón en algunas plataformas XR) para el sistema de entrada de unidad convencional. (Unity, 2020)

2.3.3.2. Asignaciones de entrada XR.

Para las entradas XR se tiene la siguiente tabla con los botones respectivos a cada uno de los dispositivos comerciales de inmersión de VR.

Cada uno de los botones tiene asignado una funcionalidad a la cual se puede acceder mediante las referencias de la siguiente tabla.

Tabla 5: Asignaciones de entrada XR

InputFeatureName	Tipo de característica	Índice de entrada (controlador izquierdo / controlador derecho)	Multi	Click	Grab/M	Empuja	OpentR (completo)	Vire	Click a través de OpentR	Multi a través de OpentR	Botón mágico
Interacción	Gr 2D	[1,2] / [4,5]	Touchpad	Palanca de mando	Touchpad	Touchpad	Touchpad / Joystick	Touchpad	Palanca de mando	Palanca de mando	Touchpad
Desmenuzador	Gr	[6,7]	Desmenuzador	Desmenuzador	Desmenuzador	Desmenuzador	Desmenuzador	Desmenuzador	Desmenuzador	Desmenuzador	Desmenuzador
Apertón	Gr	[11,12]	Apertón			Parachoques	Apertón	Apertón		Apertón	Parachoques
Desmenuzador/Click	Gr 2D	[13,14] / [15,16]	Palanca de mando							Touchpad	
Desmenuzador/Click	Botón	[18,19]	Joystick - Click								
Botón primario	Botón	[21]		[X / A] - Presione		Appt	Primario	Primario (botón estándar) (1)	Primario (X / B)	Menú	Menú
Botón secundario	Botón	[22,10]		[X / A] - Toque							
Botón de menú	Botón	[3,11]		[Y / S] - Presione			Alternar		Alternar (X / B)		
Botón de menú	Botón	[3,11]		[Y / S] - Toque							
evolución	Botón	[4,15]	Agarre - Presione	Agarre - Presione			Agarre - Presione	Agarre - Presione	Agarre - Presione	Agarre - Presione	Parachoques - Presione
Botón de menú	Botón	[5,16]	Click - Presione	Click - Presione	Click - Presione	Click - Presione	Click - Presione	Click - Presione	Click - Toque	Click - Presione	Click - Presione
Botón de menú	Botón	[6,17]	Menú	Méu (solo controlador izquierdo)							
Interacción/Click	Botón	[8,18]	Touchpad - Toque (R)	Thumbstick - Presione	Touchpad - Presione	Touchpad - Presione	Touchpad / Joystick - Presione	Touchpad - Presione	Joystick - Presione	Touchpad - Presione	
Interacción/Click	Botón	[9,17]	Touchpad - Toque	Thumbstick - Toque	Touchpad - Toque	Touchpad - Toque	Touchpad / Joystick - Toque	Touchpad - Toque	Joystick - Toque	Touchpad - Toque	Touchpad - Toque
Botón de botón	Gr		Nivel de batería								
Botón de Presión	Botón		Presencia del usuario	Presencia del usuario							

Fuente: Unity, 2020

2.3.3.3. Acceso a dispositivos de entrada.

Un InputDevice representa cualquier dispositivo físico, tal como un controlador, teléfono móvil, o el receptor de cabeza. Puede contener información sobre el seguimiento del dispositivo, botones, joysticks y otros controles de entrada. Para obtener más información sobre la InputDeviceAPI. (Unity, 2020)

Un dispositivo de entrada permanece válido a través de tramas hasta que el sistema XR lo desconecta. Use la propiedad InputDevice.IsValid para determinar si un InputDevice todavía representa un controlador activo. Puede acceder a los dispositivos de entrada de la siguiente manera:

- Características.
- Papel.
- Nodo XR.

2.3.3.4. Acceso a dispositivos de entrada por características.

Las características del dispositivo describen de qué es capaz un dispositivo o para qué se utiliza (por ejemplo, si está montado en la cabeza). Los `InputDeviceCharacteristics` es una serie de banderas se pueden agregar a su código de búsqueda de dispositivos que se ajustan a una determinada especificación. Puede filtrar dispositivos por las siguientes características:

Tabla 6: Características del dispositivo de entrada

Dispositivo	Característica
Montado en la cabeza	El dispositivo está conectado a la cabeza del usuario. Tiene seguimiento de dispositivo y seguimiento de ojo central. Este indicador se usa más comúnmente para identificar pantallas montadas en la cabeza (HMD).
Cámara	El dispositivo tiene seguimiento de cámara.
HeldInHand	El usuario sostiene el dispositivo en su mano.
HandTracking	El dispositivo representa una mano con seguimiento físico. Tiene seguimiento de dispositivo y puede contener datos de manos y huesos.
Registro visual	El dispositivo puede realizar un seguimiento ocular y tiene una función EyesData .
Dispositivo rastreado	El dispositivo se puede rastrear en el espacio 3D. Tiene dispositivo de seguimiento.
Controlador	El dispositivo tiene datos de entrada para botones y ejes, y puede usarse como controlador.
Referencia de seguimiento	El dispositivo representa un objeto de referencia de seguimiento estático. Tiene seguimiento de dispositivo, pero esos datos de seguimiento no deberían cambiar.
Izquierda	Use esta característica en combinación con las características <code>HeldInHand</code> o <code>HandTracking</code> para identificar el dispositivo como asociado con la mano izquierda.
Derecho	Use esta característica en combinación con las características <code>HeldInHand</code> o <code>HandTracking</code> para identificar el dispositivo asociado con la mano derecha.
Simulado6DOF	El dispositivo informa datos 6DOF, pero solo tiene sensores 3DOF. Unity simula los datos posicionales.

Fuente: Unity, 2020

El XR SDK subyacente informa estas características. Puede buscarlos con `InputDevice.Characteristics`. Un dispositivo puede, y a menudo debería, tener múltiples características que puede filtrar y acceder con banderas de bits. (Unity, 2020)

2.3.3.5. Acceso a dispositivos por entrada por rol

Un rol de dispositivo describe la función general de un dispositivo de entrada. Use la enumeración `InputDeviceRole` para especificar un rol de dispositivo. Los roles definidos son:

Tabla 7: Dispositivos de entrada por Rol

Papel	Descripción
Control de juego	Un controlador de juego estilo consola.
Genérico	Un dispositivo que representa el dispositivo central XR, como una pantalla montada en la cabeza o un dispositivo móvil.
HardwareTracker	Un dispositivo de rastreo.
Zurdo	Un dispositivo asociado con la mano izquierda del usuario.
Diestro	Un dispositivo asociado con la mano derecha del usuario.
Referencia de seguimiento	Un dispositivo que rastrea otros dispositivos, como una cámara de seguimiento Oculus.

Fuente: Unity, 2020

El SDK de XR subyacente informa estos roles, pero diferentes proveedores pueden organizar sus roles de dispositivo de manera diferente. Además, un usuario puede cambiar de manos, por lo que la asignación de roles podría no coincidir con la mano en la que el usuario sostiene el dispositivo de entrada. Por ejemplo, un usuario debe configurar el controlador Daydream como diestro o zurdo, pero puede optar por mantener el controlador en su mano opuesta. (Unity, 2020)

2.3.3.6. Acceso a dispositivos de entrada por nodo XR.

Los nodos XR representan los puntos de referencia físicos en el sistema XR (por ejemplo, la posición de la cabeza del usuario, sus manos derecha e izquierda, o una referencia de seguimiento como una cámara Oculus).

Tabla 8: Acceso a dispositivos de entrada por nodo XR

Nodo XR	Descripción
CenterEye	Un punto a medio camino entre las pupilas de los ojos del usuario.
Control de juego	Un controlador de juegos estilo consola. Tu aplicación puede tener múltiples dispositivos de juego.
HardwareTracker	Un dispositivo de seguimiento de hardware, generalmente conectado al usuario o un elemento físico. Pueden existir múltiples nodos de rastreador de hardware.
Cabeza	El punto central de la cabeza del usuario, calculado por el sistema XR.
Ojo izquierdo	El ojo izquierdo del usuario.
Mano izquierda	La mano izquierda del usuario.
Ojo derecho	El ojo derecho del usuario.
Mano derecha	La mano derecha del usuario.
Referencia de seguimiento	Un punto de referencia de seguimiento, como la cámara Oculus. Pueden existir múltiples nodos de referencia de seguimiento.

Fuente: Unity, 2020

2.3.3.7. Escuchando las conexiones y desconexiones del dispositivo.

Los dispositivos de entrada son consistentes de cuadro a cuadro, pero pueden conectarse o desconectarse en cualquier momento. Para evitar comprobar repetidamente si un dispositivo está conectado a la plataforma, use `InputDevices.deviceConnected` y `InputDevices.deviceDisconnected` para notificar a

su aplicación cuando un dispositivo se conecta o desconecta. Estos también le proporcionan una referencia al dispositivo de entrada recién conectado.

Como puede retener estas referencias en varios marcos, un dispositivo puede desconectarse o dejar de estar disponible. Para verificar si las entradas de un dispositivo todavía están disponibles, use `InputDevice.isValid`. Scripts que los dispositivos de entrada de acceso deben verificar esto al comienzo de cada cuadro antes de intentar usar ese dispositivo. (Unity, 2020)

2.3.3.8. Acceso a datos de seguimiento manual.

`InputDevices` admite dispositivos de seguimiento manual. Un dispositivo de seguimiento manual siempre:

- Tiene la característica `HandTracking` contiene un uso de `CommonUsages.HandData` del tipo `Hand`.
- Los datos de seguimiento manual consisten en un objeto manual y una serie de hasta 21 características de entrada de hueso. Cada hueso tiene una posición y orientación, así como referencias a sus huesos primarios y secundarios en la jerarquía. El objeto `Mano` puede obtener el hueso raíz o una lista de huesos para cada dedo individual.
- Cuando `Hand.tryGetRootBone` obtiene el hueso raíz, recupera un objeto que representa un hueso ubicado justo encima de la muñeca. También puede obtener una lista de huesos que representa cada dedo individual. Calling `Hand.TryGetFingerBones` devuelve una lista, desde el nudillo hasta la punta, de los huesos que representan ese dedo.

2.3.3.9. Acceso a datos de seguimiento ocular.

Los dispositivos de entrada admiten dispositivos de seguimiento ocular, así como dispositivos de seguimiento manual. El seguimiento ocular consiste en las posiciones de los ojos izquierdo y derecho, la ubicación en el espacio 3D donde mira el usuario y la cantidad de parpadeo de cada ojo individual. Su tipo de datos es `Ojos`. Para recuperarlo de un dispositivo, use `CommonUsages.eyesData`. (Unity, 2020)

2.3.3.10. Asociación `XRInputSubsystem` y `InputDevice`.

Unity proporciona dos sistemas de entrada: el sistema de entrada heredado y la arquitectura del complemento XR introducida en 2019.2. En la nueva configuración, cada `InputDevice` está asociado con un `XRInputSubsystem`. Estos objetos del subsistema controlan el comportamiento de entrada global que no está asociado con ningún dispositivo de entrada específico (por ejemplo, administrar el origen de seguimiento o volver a centrar los dispositivos rastreados).

Cada `InputDevice` contiene una referencia a su subsistema asociado. Esta referencia es nula si el dispositivo proviene de una plataforma integrada. También puede obtener todos los objetos `XRInputSubsystem` activos con `SubsystemManager.GetInstances <XRInputSubsystem>`, y cada `XRInputSubsystem` puede obtener sus dispositivos con `XRInputSubsystem.TryGetInputDevices`.

Puede usar el Subsistema de entrada para volver a centrar dispositivos con `UnityEngine.XR.XRInputSubsystem.Recenter`, establece la posición actual del HMD como el nuevo origen para todos los dispositivos. Devuelve falso para los dispositivos que no se pueden volver a ingresar, o si la plataforma no admite la reincorporación. Para recuperar el límite de seguimiento, use `TryGetBoundaryPoints`. Esto consiste en una serie de puntos 3D ordenados en el sentido de las agujas del reloj, donde el valor y está a nivel del piso, y marcan la 'zona segura' especificada por el usuario para colocar contenido e interacciones. Puede escuchar los cambios en este límite con `XRInputSubsystem.boundaryChanged`.

El `XRInputSubsystem` también es responsable del modo de origen de seguimiento, que proporciona el contexto de dónde está el origen del mundo de seguimiento.

Unity admite los siguientes modos de origen de seguimiento:

- Dispositivo: la ubicación del origen es la primera ubicación conocida del dispositivo de visualización principal; a menudo un HMD o un teléfono.
- Piso: la ubicación del origen está en un lugar conocido en el piso.
- Referencia de seguimiento: la ubicación del origen está en un dispositivo de entrada con el conjunto de características de referencia de seguimiento.
- Desconocido: el tipo de origen de seguimiento es desconocido. Esto puede deberse a una falla del sistema o la falta de seguimiento del soporte en modo de origen.

Hay tres API que puede usar para administrar el modo de origen de seguimiento:

- `XRInputSubsystem.CalculateSetTrackingOriginMode`: establece el modo de origen de seguimiento.
- `XRInputSubsystem.GetTrackingOriginMode`: recupera el modo de origen de seguimiento.
- `XRInputSubsystem.GetSupportedTrackingOriginModes`: recupera todos los modos de origen de seguimiento que admite el SDK.

Entrada XR a través del sistema de entrada heredado.

Todavía puede usar el sistema de entrada heredado, que consiste en `Input` y `XR.InputTracking`, para recuperar las funciones de entrada XR. Para hacer esto, use los índices de entrada heredados apropiados de la tabla de asignaciones de entrada XR en esta página. En la sección de entrada de la configuración del jugador (menú: Edición > Configuración del proyecto > Entrada), cree una asignación de eje para agregar la asignación apropiada del nombre de entrada al índice del eje para la función del dispositivo de la plataforma. Para recuperar el botón o el valor del eje, use `Input.GetAxisInput.GetButton` y pase el eje ahora asignado o el nombre del botón.

Para obtener más información sobre cómo usar los botones y los ejes del joystick. (Unity, 2020)

2.3.4. Físicas.

Un cuerpo rígido es el componente principal que permite el comportamiento físico de un `GameObject` con un cuerpo rígido conectado, el objeto responderá inmediatamente a la gravedad, si uno o más `Colliders` También se agregan componentes, el `GameObject` es movido por colisiones entrantes.

Dado que un componente `Rigidbody` se hace cargo del movimiento del `GameObject` al que está adjunto, no debe intentar moverlo desde un script cambiando las propiedades de `Transform`, como la posición y la rotación. En cambio, debe aplicar fuerzas para empujar el `GameObject` y dejar que el motor físico sea calcular los resultados.

Hay algunos casos en los que es posible que desee que un `GameObject` tenga un cuerpo rígido sin que su movimiento sea controlado por el motor de física. Por

ejemplo, es posible que desee controlar su personaje directamente desde el código del script, pero aun así permitir que sea detectado por los desencadenantes. Este tipo de movimiento no físico producido a partir de un guión se conoce como movimiento cinemático. El componente Rigidbody tiene una propiedad llamada `Is Kinematic` que lo elimina del control del motor de física y permite que se mueva cinemáticamente desde un script. Es posible cambiar el valor de `Is Kinematic` desde un script para permitir que la física se active y desactive para un objeto, pero esto viene con una sobrecarga de rendimiento y debe usarse con moderación. (Unity, 2020)

2.3.4.1. Dormido.

Cuando un cuerpo rígido se mueve más lento que una velocidad lineal o rotacional mínima definida, el motor de física asume que se ha detenido. Cuando esto sucede, el `GameObject` no se mueve de nuevo hasta que recibe una colisión o fuerza, por lo que se establece en modo de "suspensión". Esta optimización significa que no se gasta tiempo de procesador actualizando el Rigidbody hasta la próxima vez que se "despierte" (es decir, se ponga en movimiento nuevamente).

Para la mayoría de los propósitos, dormir y despertar de un componente Rigidbody ocurre de manera transparente. Sin embargo, un `GameObject` podría no despertarse si un Colisionador estático (es decir, uno sin un Cuerpo rígido) se mueve hacia adentro o hacia afuera modificando la posición `Transform`. Esto podría resultar, por ejemplo, en el Rigidbody `GameObject` que cuelga en el aire cuando el piso se ha movido desde debajo. En casos como este, el `GameObject` se puede despertar explícitamente usando la `WakeUp` función. (Unity,2020)

2.3.4.2. Colisionadores.

Los componentes del colisionador definen la forma de un `GameObject` para colisiones físicas. Un colisionador, que es invisible, no necesita tener exactamente la misma forma que la malla de `GameObject`. Una aproximación aproximada de la malla a menudo es más eficiente e indistinguible en el juego.

Los colisionadores más simples (y menos intensivos en procesador) son los tipos de colisionadores primitivos.

En 3D, estos son el Box Collider, Sphere Collider y Capsule Collider. En 2D, puede usar Box Collider 2D y Circle Collider 2D. Puede agregar cualquier número de estos a un solo GameObject para crear colisionadores compuestos. (Unity, 2020)

2.3.4.3. Colisionadores compuestos.

Los colisionadores compuestos se aproximan a la forma de un GameObject mientras mantienen una baja sobrecarga del procesador. Para obtener mayor flexibilidad, puede agregar colisionadores adicionales en GameObjects secundarios. Por ejemplo, puede rotar cuadros en relación con los ejes locales del GameObject padre. Cuando crea un colisionador compuesto como este, solo debe usar un componente Rigidbody, colocado en el GameObject raíz en la jerarquía.

Los colisionadores primitivos no funcionan correctamente con las transformaciones de corte. Si utiliza una combinación de rotaciones y escalas no uniformes en la jerarquía de transformación para que la forma resultante ya no sea una forma primitiva, el colisionador primitivo no puede representarla correctamente. (Unity, 2020)

2.3.4.4. Colisionadores de malla.

Sin embargo, hay algunos casos en los que incluso los colisionadores compuestos no son lo suficientemente precisos. En 3D, puede usar Mesh Colliders para que coincida exactamente con la forma de la malla de GameObject. En 2D, Polygon Collider 2D no coincide perfectamente con la forma del gráfico de sprite, pero puede refinar la forma a cualquier nivel de detalle que desee.

Estos colisionadores son mucho más intensivos en el procesador que los tipos primitivos, así que úselos con moderación para mantener un buen rendimiento. Además, un colisionador de malla no puede chocar con otro colisionador de malla (es decir, no sucede nada cuando hacen contacto). Puede solucionar esto en algunos casos marcando el colisionador de malla como Convexo en el Inspector. Esto genera la forma del colisionador como un "casco convexo" que es como la malla original, pero con cualquier socavado relleno.

El beneficio de esto es que un colisionador de malla convexo puede colisionar con otros colisionadores de malla, por lo que puede usar esta función cuando tenga un personaje en movimiento con una forma adecuada. Sin embargo, una buena regla es usar colisionadores de malla para la geometría de la escena y aproximar la forma de los GameObjects en movimiento usando colisionadores primitivos compuestos. (Unity,2020)

2.3.4.5. Colisionadores estáticos.

Puede agregar colisionadores a un GameObject sin un componente Rigidbody para crear pisos, paredes y otros elementos inmóviles de una escena. Estos se conocen como colisionadores estáticos. No debe reposicionar los colisionadores estáticos cambiando la posición de Transformar porque esto impacta fuertemente en el rendimiento del motor de física.

Los colisionadores en un GameObject que tiene un cuerpo rígido se conocen como colisionadores dinámicos. Los colisionadores estáticos pueden interactuar con colisionadores dinámicos, pero como no tienen un cuerpo rígido, no se mueven en respuesta a colisiones. (Unity, 2020)

2.3.4.6. Materiales de física.

Cuando los colisionadores interactúan, sus superficies necesitan simular las propiedades del material que se supone que representan. Por ejemplo, una capa de hielo será resbaladiza, mientras que una pelota de goma ofrecerá mucha fricción y será muy hinchable. Aunque la forma de los colisionadores no se deforma durante las colisiones, su fricción y rebote se pueden configurar utilizando materiales de física. Obtener los parámetros correctos puede implicar un poco de prueba y error, pero un material de hielo, por ejemplo, tendrá una fricción cero (o muy baja) y un material de goma con alta fricción y un rebote casi perfecto. (Unity, 2020)

2.3.4.7. Disparadores.

El sistema de secuencias de comandos puede detectar cuándo se producen colisiones e iniciar acciones mediante la función OnCollisionEnter. Sin embargo, también puede usar el motor de física simplemente para detectar cuándo un

colisionador ingresa al espacio de otro sin crear una colisión. Un colisionador configurado como Trigger (usando la propiedad Is Trigger) no se comporta como un objeto sólido y simplemente permitirá que pasen otros colisionadores. Cuando un colisionador ingresa a su espacio, un disparador llamará a la función OnTriggerEnter en los scripts del objeto disparador. (Unity, 2020)

2.3.4.8. Devolución de llamadas de colisión para scripts.

Cuando se producen colisiones, el motor de física llama a funciones con nombres específicos en cualquier script adjunto a los objetos involucrados. Puede colocar cualquier código que desee en estas funciones para responder al evento de colisión. Por ejemplo, puede reproducir un efecto de sonido de choque cuando un automóvil choca con un obstáculo. En la primera actualización física donde se detecta la colisión, se llama a la función OnCollisionEnter. Durante las actualizaciones donde se mantiene el contacto, se llama a OnCollisionStay y finalmente, OnCollisionExit indica que el contacto se ha interrumpido.

Tabla 9: Se produce la detección de colisión y se envían mensajes tras la colisión.

Collision detection occurs and messages are sent upon collision						
	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider		Y				
Rigidbody Collider	Y	Y	Y			
Kinematic Rigidbody Collider		Y				
Static Trigger Collider						
Rigidbody Trigger Collider						
Kinematic Rigidbody Trigger Collider						

Fuente: Unity, 2020

Tabla 10: Los mensajes de activación se envían en caso de colisión.

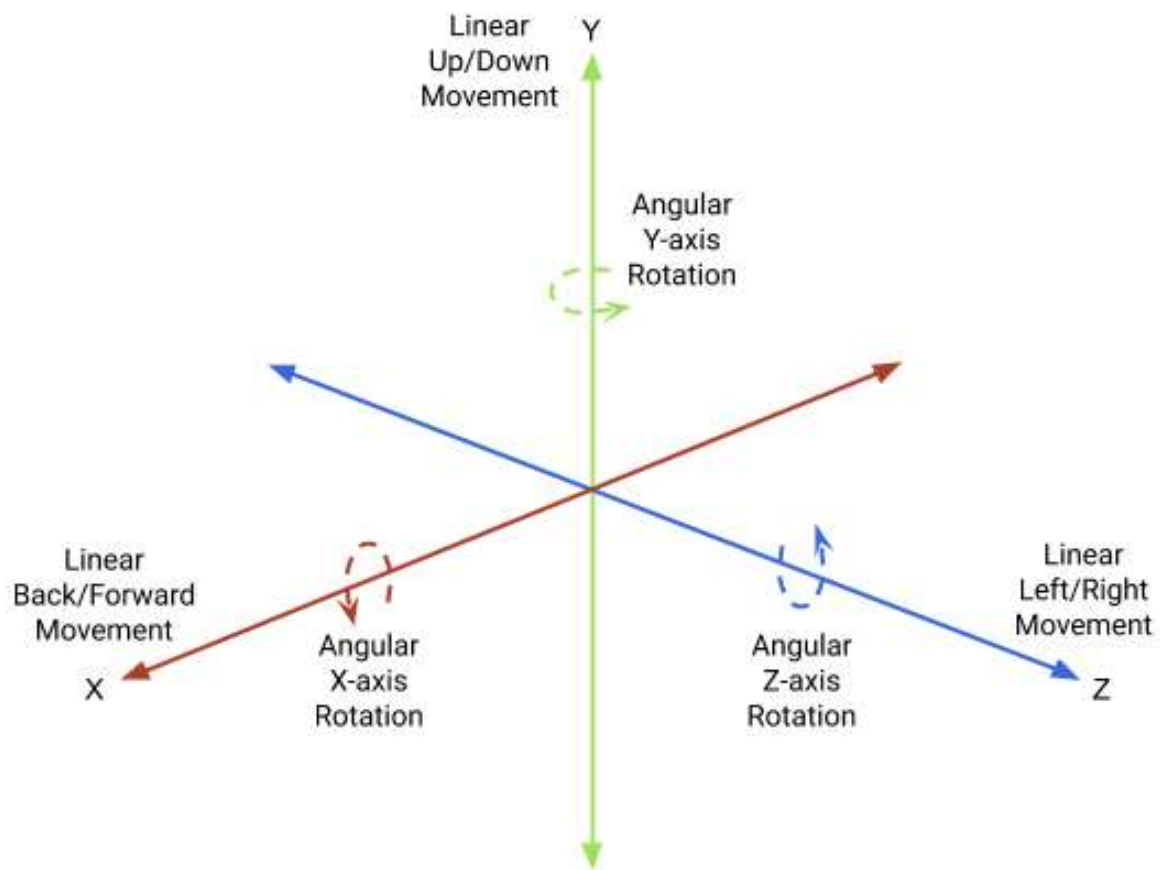
Trigger messages are sent upon collision						
	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider					Y	Y
Rigidbody Collider				Y	Y	Y
Kinematic Rigidbody Collider				Y	Y	Y
Static Trigger Collider		Y	Y		Y	Y
Rigidbody Trigger Collider	Y	Y	Y	Y	Y	Y
Kinematic Rigidbody Trigger Collider	Y	Y	Y	Y	Y	Y

Fuente: Unity, 2020

2.3.4.9. Articulaciones.

Un componente conjunto conecta un cuerpo rígido a otro cuerpo rígido o un punto fijo en el espacio. Las articulaciones aplican fuerzas que mueven cuerpos rígidos, y los límites de las articulaciones restringen ese movimiento. Las articulaciones otorgan a los cuerpos rígidos los siguientes grados de libertad:

Figura 26: Articulaciones



Fuente: Unity, 2020

Unity proporciona las siguientes articulaciones que aplican diferentes fuerzas y límites a los componentes del cuerpo rígido y, por lo tanto, les dan a esos cuerpos un movimiento diferente:

Tabla 11: Tabla de articulaciones.

Property:	Function:
<u>Character Joint</u>	Emulates a ball and socket joint, like a hip or shoulder. Constrains rigid body movement along all linear degrees of freedom, and enables all angular freedoms. Rigidbodies attached to a Character Joint orient around each axis and pivot from a shared origin.
<u>Configurable Joint</u>	Emulates any skeletal joint, like those in a ragdoll. You can configure this joint to force and restrict rigid body movement in any degree of freedom.
<u>Fixed Joint</u>	Restricts the movement of a rigid body to follow the movement of the rigid body it is attached to. This is useful when you need rigid bodies that easily break apart from each other, or you want to connect the movement of two rigid bodies without parenting in a <u>Transform</u> hierarchy.
<u>Hinge Joint</u>	Attaches a rigid body to another rigid body or a point in space at a shared origin and allows the rigid bodies to rotate around a specific axis from that origin. Useful for emulating doors and finger joints.
<u>Spring Joint</u>	Keeps rigid bodies apart from each other but lets the distance between them stretch slightly. The spring acts like a piece of elastic that tries to pull the two anchor points together to the exact same position.

Fuente: Unity, 2020

Las articulaciones también tienen otras opciones que puede habilitar para efectos específicos. Por ejemplo, puede configurar una unión para que se rompa cuando un cuerpo rígido le aplica una fuerza que excede un cierto umbral. Algunas articulaciones permiten que se produzca una fuerza impulsora entre los cuerpos rígidos conectados para ponerlos en movimiento automáticamente. (Unity, 2020)

2.3.5. Scripting.

La creación de secuencias de comandos es un ingrediente esencial en todas las aplicaciones que realice en Unity. La mayoría de las aplicaciones necesitan scripts para responder a las aportaciones del jugador y organizar los eventos en el juego para que sucedan cuando deberían. Más allá de eso, los scripts se pueden usar para crear efectos gráficos, controlar el comportamiento físico de los objetos o incluso implementar un sistema de IA personalizado para los personajes del juego. (Unity, 2020)

2.3.5.1. Scripting Overview.

Si bien Unity utiliza una implementación del tiempo de ejecución estándar de Mono para los scripts, todavía tiene sus propias prácticas y técnicas para acceder al motor desde los scripts. (Unity, 2020)

2.3.5.2. Crear y usar scripts.

El comportamiento de `GameObject` está controlado por los componentes que están unidos a ellos. Aunque los componentes integrados de Unity pueden ser muy versátiles, pronto descubrirá que necesita ir más allá de lo que pueden proporcionar para implementar sus propias funciones de juego. Unity le permite crear sus propios componentes utilizando scripts. Estos le permiten activar eventos del juego, modificar las propiedades de los componentes a lo largo del tiempo y responder a la entrada del usuario de la forma que desee.

Unity admite el lenguaje de programación C # de forma nativa. C # (pronunciado C-sharp) es un lenguaje estándar de la industria similar a Java o C ++.

Aprender el arte de la programación y el uso de estos lenguajes particulares está más allá del alcance de esta introducción. (Unity, 2020)

2.3.5.3. Crear guiones.

A diferencia de la mayoría de los otros activos, los scripts generalmente se crean dentro de Unity directamente. Puede crear un nuevo script desde el menú Crear en la esquina superior izquierda del panel Proyecto o seleccionando Activos > Crear > Script C # en el menú principal.

El nuevo script se creará en la carpeta que haya seleccionado en el panel Proyecto. Se seleccionará el nombre del nuevo archivo de script, que le solicitará que ingrese un nuevo nombre. (Unity, 2020)

2.3.5.4. Variables y el inspector.

Al crear un script, básicamente está creando su propio nuevo tipo de componente que se puede conectar a Game Objects como cualquier otro componente. (Unity, 2020)

2.3.5.5. Atributos.

Los atributos son marcadores que se pueden colocar sobre una clase, propiedad o función en un script para indicar un comportamiento especial. Por ejemplo, puede agregar el atributo `HideInInspector` encima de una declaración de propiedad para evitar que el Inspector muestre la propiedad, incluso si es pública. C # contiene nombres de atributos entre corchetes. (Unity, 2020)

2.3.6. Audio.

Un juego estaría incompleto sin algún tipo de audio, ya sea música de fondo o efectos de sonido. El sistema de audio de Unity es flexible y potente. Puede importar la mayoría de los formatos de archivo de audio estándar y tiene características sofisticadas para reproducir sonidos en el espacio 3D, opcionalmente con efectos como eco y filtrado aplicados. Unity también puede grabar audio desde cualquier micrófono disponible en la máquina de un usuario para usar durante el juego o para almacenamiento y transmisión. En la vida real, los objetos emiten sonidos y los oyentes los escuchan. La forma en que se percibe un sonido depende de varios factores. Un oyente puede decir aproximadamente de qué dirección proviene un sonido y también puede tener una idea de su distancia de su volumen y calidad. Una fuente de sonido de rápido movimiento (como una bomba que cae o un automóvil de policía que pasa) cambiará de tono a medida que se mueva como resultado del efecto Doppler. Además, los alrededores afectarán la forma en que se refleja el sonido, por lo que una voz dentro de una cueva tendrá un eco, pero la misma voz al aire libre no.

Figura 27: Sorround



Fuente: Unity, 2020

Para simular los efectos de la posición, Unity requiere que los sonidos se originen a partir de fuentes de audio adjunto a otro objeto, la mayoría de las veces la cámara principal. Unity puede simular los efectos de la distancia y la posición de una fuente desde el objeto del oyente y reproducirlos para el usuario en consecuencia. La

velocidad relativa de los objetos fuente y de escucha también se puede utilizar para simular el efecto Doppler para un mayor realismo.

La unidad no puede calcular ecos puramente de la escena geometría, pero puede simularlos agregando filtros de audio a los objetos. Por ejemplo, podría aplicar el filtro Echo a un sonido que se supone que proviene del interior de una cueva. En situaciones donde los objetos pueden moverse dentro y fuera de un lugar con un fuerte eco, puede agregar una zona de reverberación a la escena. Por ejemplo, su juego podría involucrar autos que circulan por un túnel. Si coloca una zona de reverberación dentro del túnel, los sonidos del motor de los automóviles comenzarán a hacer eco cuando entren y el eco se apagará cuando emerjan del otro lado. (Unity, 2020)

2.3.6.1. Interfaces de usuario (UI).

UIElements: User Interface Elements (UIElements) es un kit de herramientas de IU en modo retenido para desarrollar interfaces de usuario en el Editor de Unity. UIElements se basa en tecnologías web reconocidas y admite hojas de estilo, manejo de eventos dinámicos y contextuales y persistencia de datos.

Unity UI (paquete): el paquete Unity User Interface (Unity UI) proporciona un kit de herramientas de UI simple para desarrollar interfaces de usuario para juegos y aplicaciones. Unity UI es un sistema de interfaz de usuario basado en GameObject que utiliza componentes y Game View para organizar, posicionar y diseñar la interfaz de usuario. No puede usar la interfaz de usuario de Unity para interfaces de usuario dentro del Editor de Unity.

ImGui: la interfaz gráfica de usuario de modo inmediato es un kit de herramientas de interfaz de usuario basado en código que está destinado principalmente como una herramienta para desarrolladores. ImGui utiliza la función OnGUI (y scripts que implementan la función OnGUI) para dibujar y administrar su interfaz de usuario. Puedes usar ImGui para crear pantallas de depuración en el juego, inspectores personalizado para componentes de script y ventanas o herramientas que amplían el

Editor de Unity. No es la mejor opción para crear la interfaz de usuario de tu juego o aplicación. (Unity, 2020)

2.4. MODELADO 3D CON BLENDER.

Blender es un software 3D todo en uno que se puede usar para modelar, esculpir, texturizar, animar, rastrear, renderizar y componer gráficos de aspecto increíble de principio a fin. ¿La mejor parte? ¡Es gratis para todos!

Gratis a menudo significa baja calidad, pero afortunadamente ese no es el caso con Blender. Fue escrito en 1995 por Ton Roosendaal como un software interno para un estudio de animación holandés llamado NeoGeo. Originalmente se vendió a otros estudios también, pero a principios de 2002 Blender necesitaba ser archivado debido al declive económico. Para mantener vivo su proyecto, Roosendaal convenció a los inversores de convertir Blender en una Licencia Pública General GNU recaudando 100.000 € en solo siete semanas. Blender ha sido gratuito desde entonces, y se está desarrollando continuamente gracias a las generosas donaciones de la comunidad. (Lampel, 2015, pp: 4)

2.4.1. Bloqueando.

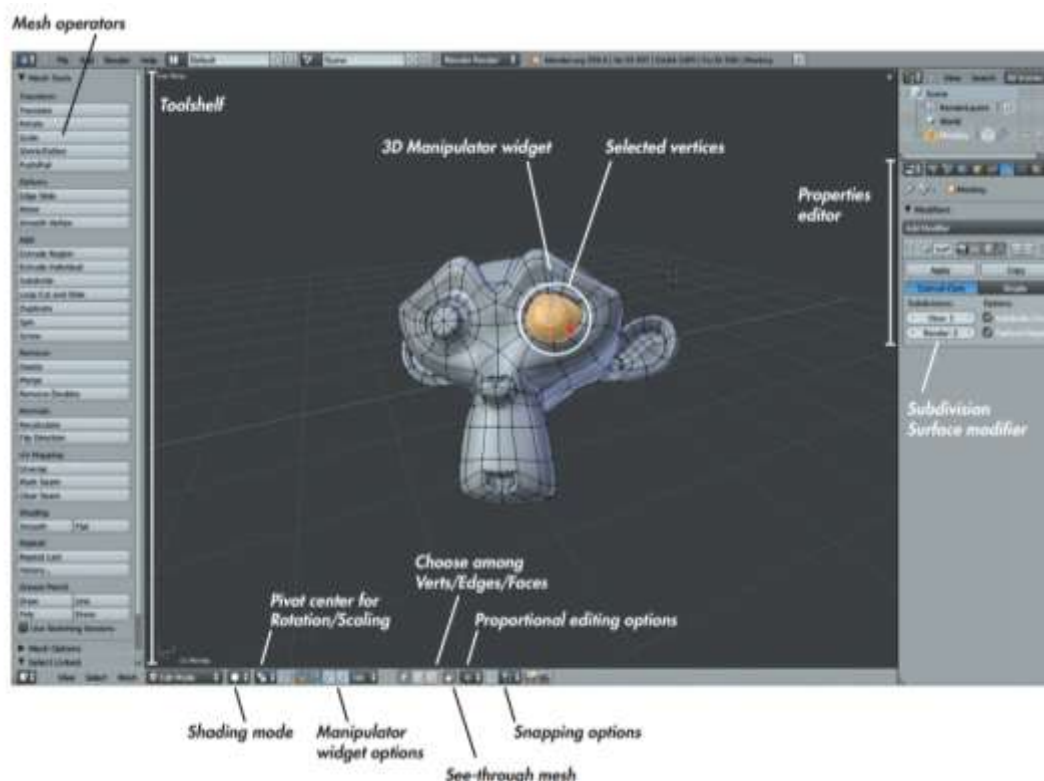
El modelado puede ser un proceso largo y, para ahorrar tiempo, es útil saber a dónde va con una tarea de modelado en particular. Aún así, no puede modelar un objeto de una vez; debe comenzar en algún lugar, y ahí es donde entra el bloqueo. El bloqueo es el proceso de agregar marcadores de posición simples para los objetos que desea crear para que pueda planificar mejor cómo encajan, considerar sus construcciones individuales y detectar problemas antes se convierten en problemas importantes. Una vez que se ha bloqueado una escena, puede pasar a refinar, reemplazar o agregar a cada parte para crear un modelo final. Por ejemplo, en la escena del Templo de la Selva, elementos simples como cubos se usan como marcadores de posición para elementos clave; para otros proyectos, creamos mallas

base simples para esculpir que actúan como base para desarrollar la forma final de nuestros modelos esculpidos.

2.4.1.1. Modo de edición.

El modo de edición es donde ocurre el modelado. Con un objeto de malla seleccionado, puede ingresar al modo Editar presionando la pestaña o haciendo clic en el menú desplegable del modo en el encabezado de la Vista 3D y seleccionando el modo Editar. Una vez en el modo Edición, el objeto seleccionado es editable (si de hecho es editable, los objetos vacíos y las lámparas, por ejemplo, no lo son). Cuando un objeto es editable, puede seleccionar y manipular partes, así como crear nuevas partes. (Simons, 2013, pp: 29)

Figura 28: Modo edición



Fuente: Simons, 2013

2.4.1.2. Modificadores.

Los modificadores le permiten realizar operaciones procesales y no destructivas en sus modelos. Son fundamentales para crear casi cualquier cosa en Blender. Algunos modificadores generan nueva geometría, reemplazando o agregando a su malla, mientras que otros deforman la geometría existente de acuerdo con ciertas reglas u ofrecen formas de conectar simulaciones y otras entidades más complejas en su escena. (Simons, 2013, pp: 29)

Figura 29: Modificadores



Fuente: Simons, 2013

2.4.1.3. Snapping.

Blender tiene herramientas para ajustar objetos, vértices, caras o bordes a todo tipo de cosas. Para activar Snapping, haga clic en el ícono Snapping (el imán de herradura) en el encabezado de la vista 3D. Luego, desde el menú desplegable junto al ícono Ajustar, seleccione cómo Blender ajustará su selección cuando se mueva, escale o rote entre las siguientes opciones: Incremento Su selección se ajustará al incremento más cercano para que pueda construir objetos con puntos alineados con precisión. Esto es útil para modelar cosas como edificios u objetos mecánicos donde desea paredes y pisos perfectamente alineados sin grumos ni protuberancias. (También puede ajustar su selección a la cuadrícula de Blender presionando shift-S4Selection a Cuadrícula). Vértice / Borde / Cara / Volumen Su selección se ajustará a los vértices, bordes, caras o interior de cualquier objeto. Puedes cambiar lo que Blender elige ajustar desde el menú desplegable que aparece. Haga clic en el icono a la derecha del menú desplegable para rotar la selección de modo que se alinee

con los vértices normales a los que se ajusta. Una opción de ajuste muy importante se encuentra en el modo Ajuste de rostro. Al habilitar el ajuste de cara, aparece el icono Proyecto en superficie. Proyectar en superficie hará que la geometría que cree se ajuste a la superficie de los objetos existentes a medida que los mueva, escale o rote. Esta opción le permite crear una nueva topología sobre la superficie de un objeto existente. Crearemos una nueva topología sobre la superficie de los objetos existentes para capturar mejor sus formas después de haber esculpido los originales.

Mallas base Para crear la Criatura Bat, necesitaba una malla base simple que pudiera esculpirse. Una malla base es un modelo simple que captura las formas básicas del modelo que desea esculpir. Una vez que haya creado una malla base, puede agregar un modificador Multiresolución y comenzar a subdividir y esculpir detalles en el modo Esculpir. Su malla base debe capturar las proporciones generales de la malla y estar diseñada para subdividirse fácilmente con el fin de proporcionar una buena malla uniforme para esculpir. Podemos retopologizar la malla base esculpida más tarde para crear el modelo final, pero trabajar con una geometría muy simple ahora dejará más libertad para la experimentación mientras esculpe. Una malla base puede tener cualquier nivel de complejidad, pero intente crear una topología que admita los formularios que sabe que desea crear, sin introducir demasiados detalles de los que podría no estar seguro. Dependiendo de qué tan seguro esté de cómo quiere que se vea su modelo, puede incluir topología para características como ojos, bocas y grupos musculares, o puede dejarlo tan simple como una esfera o cubo básico si solo está modelando una cara. Para Bat Creature, apunté a una malla base que proporcionara topología para el plan general del cuerpo, pero omitió detalles finos, como la cabeza. Desarrollé la malla base estableciendo mi arte conceptual como imagen de fondo y utilizándolo como guía. (Simons, 2013, pp: 55)

2.4.1.4. Esculpir.

Modo Esculpir Para esculpir en Blender, deberás ingresar al modo Esculpir en la vista 3D. Una vez en el modo Esculpir, asegúrate de tener el estante de herramientas abierto (T), ya que aquí es donde encontrarás todo el esculpido de Blender opciones de modo y herramientas. El familiar Estante de herramientas a la

izquierda de la ventana gráfica ahora alberga nuestras opciones de Pincel para esculpir, donde podemos elegir entre diferentes pinceles y ajustar su configuración. Las opciones de pincel incluyen el tipo de pincel; si utilizará la entrada de presión de una tableta gráfica (se recomienda si tiene una); y el tamaño, la caída y la forma del pincel. La Figura 30 también muestra cómo se configuró la Vista 3D para facilitar el esculpido: puede usar el sombreado GLSL de Blender y habilitar Solo Renderizar para mostrar solo los objetos que se renderizarán. Esto mantiene la cuadrícula y cualquier objeto superfluo en su escena, como lámparas y cámaras, fuera de su camino mientras esculpe. Cubriremos estos ajustes con más profundidad en "Personalización de la vista 3D".

Figura 30: Personalización de la vista 3D



Fuente: Simons, 2013

2.4.2. Horneado de texturas.

Después de modelar, esculpir, retopologizar y desenvolver con UV nuestros modelos, se realizará la creación de texturas y materiales para ellos y luego pasar a la iluminación y el renderizado. En cuanto a Texturas como estas se pueden usar para ayudar a texturizar y crear materiales más adelante. (Simons, 2013, pp: 137-139)

Para hornear una textura (técnicamente, una imagen, de aquí en adelante usaré los dos términos con menos rigor) en Blender, use el panel Bake en la pestaña Render del editor de Propiedades. Contiene todas las configuraciones y herramientas de Blender para hornear mapas de texturas, incluidas las siguientes:

Hornear: Este botón hornea imágenes para los objetos seleccionados actualmente, utilizando la configuración de horneado que ha definido.

Modo de horneado: Esta configuración se usa para establecer qué tipo de mapa está horneando. Borrar Esta opción borra la textura y la reemplaza con negro antes de hornear.

Margen: Esta opción extiende las texturas horneadas más allá de los bordes de las islas UV en la cantidad de píxeles que establezca para evitar que se vean las costuras en la malla.

Seleccionado: como Activo Esta opción permite hornear de una malla a otra.

Distancia y sesgo Estas configuraciones determinan qué tan lejos Blender buscará la superficie de la otra malla cuando hornee de Seleccionado a Activo. Para hornear una textura, Blender necesita dos entradas:

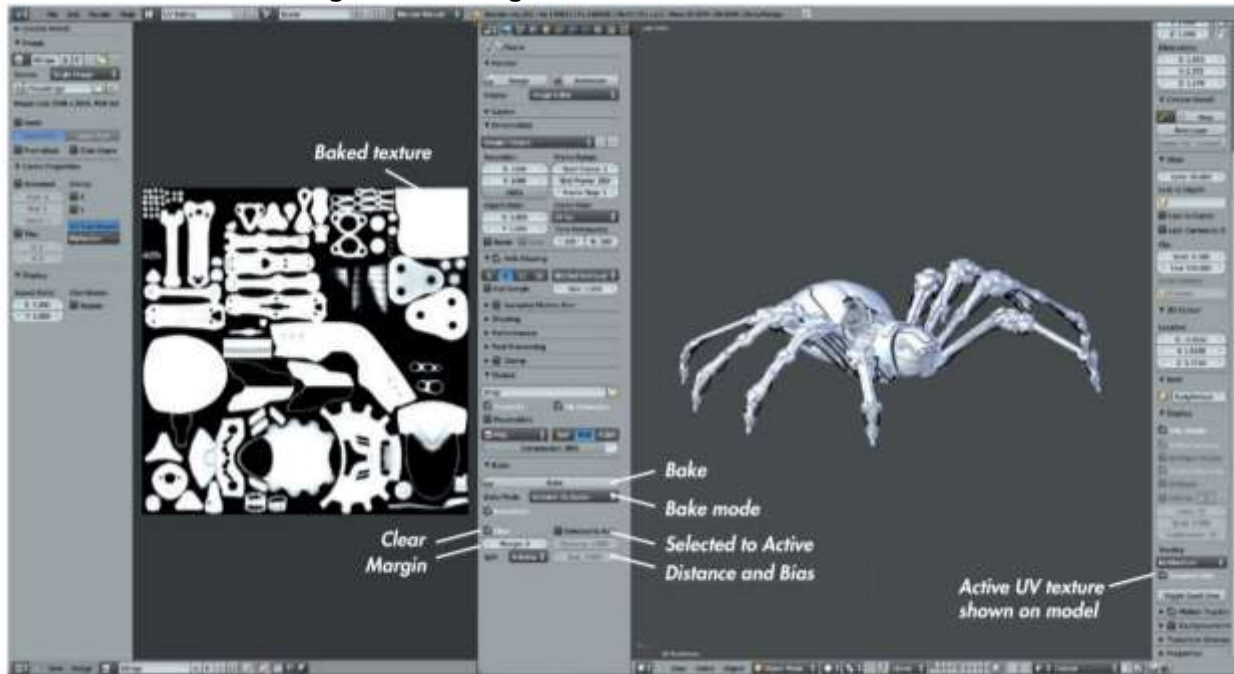
- Una malla con coordenadas UV para la que desea hornear texturas. Esto debe seleccionarse como el objeto activo, y las coordenadas UV que desea hornear deben ser el conjunto activo.
- Una imagen para hornear la textura, asignada al conjunto de coordenadas UV activo del objeto. Con estas entradas en su lugar, haga clic en Hornear en el panel Hornear para que Blender procese la textura.

Blender debe representar el tipo de mapa dictado por la configuración del modo de horneado (ver Figura 31) y tener en cuenta lo siguiente al hornear:

Objetos seleccionados Blender horneará texturas para cualquier objeto seleccionado que tenga coordenadas UV con una imagen asignada a ellos. Puede hornear múltiples texturas a la vez, así como hornear múltiples objetos.

Otros objetos en las capas actualmente visibles Los objetos no seleccionados no se hornearán, sino al hornear luces u otros mapas afectados.

Figura 31: Configuración del modo de horneado



Fuente: Simons, 2013

2.4.2.1. Tipos de mapas de textura.

Puede hornear varios tipos de mapas diferentes, algunos de los cuales son más importantes que otros. Esto es lo que Blender tiene para ofrecer:

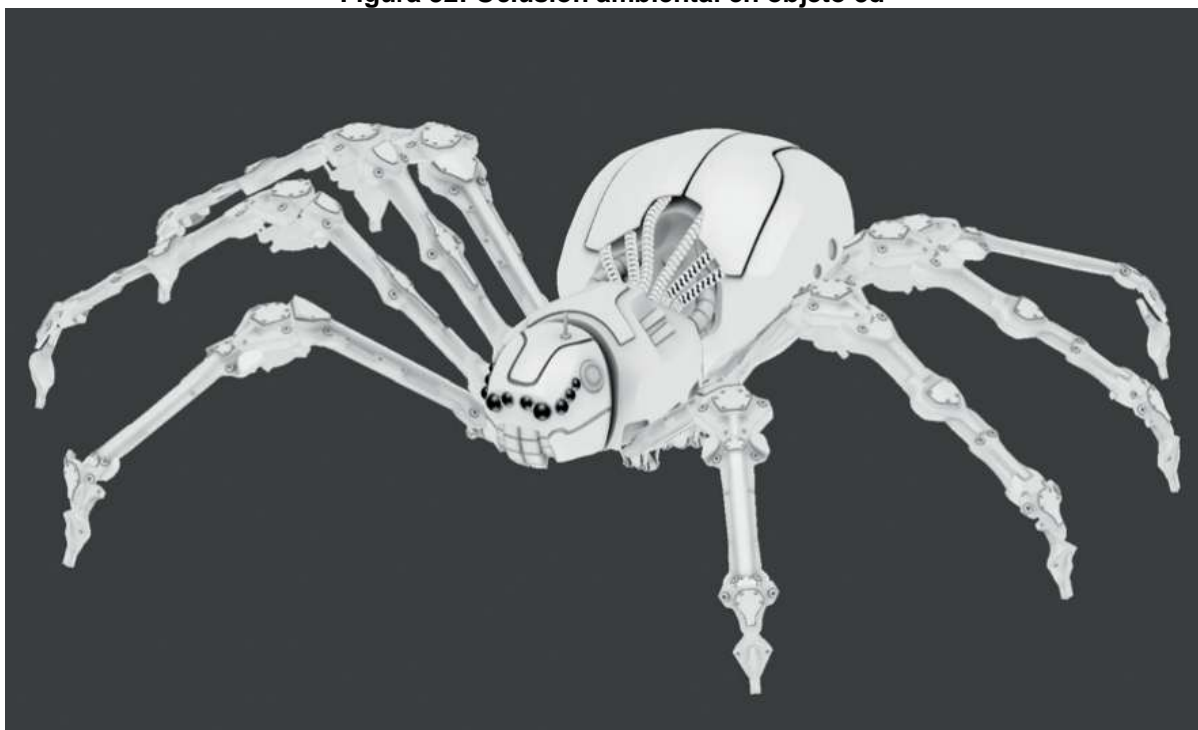
Renderizado completo Realiza un renderizado completo, que incluye texturas e iluminación de su superficie. La textura resultante es exactamente como se verá su modelo en su render final, mapeado a las coordenadas UV del objeto.

Oclusión ambiental Representa la oclusión ambiental del objeto, teniendo en cuenta cualquier otro objeto visible y renderizable en las capas activas actualmente. La oclusión ambiental es un efecto de auto sombra que crea áreas oscuras en las esquinas y grietas de los objetos. Es útil tanto para imitar el efecto de una iluminación más compleja como para crear un mapa aproximado de áreas donde es probable que se acumule suciedad y polvo.

Sombra Esto convierte las sombras de las luces en la escena en un mapa.

Normals Esto toma el vector normal de la superficie del objeto en cada punto y lo registra como un valor RGB. Esto se realiza de varias maneras, dependiendo de la opción de espacio normal que elija. Diferentes espacios normales registran las normales de la malla en relación con diferentes sistemas de coordenadas. Por ejemplo, la opción Cámara los graba en relación con la cámara; Mundo, según el espacio mundial; Objeto, de acuerdo con las coordenadas locales del objeto; y Tangente, de acuerdo con la superficie normal del objeto horneado.

Figura 32: Oclusión ambiental en objeto 3d



Fuente: Simons, 2013

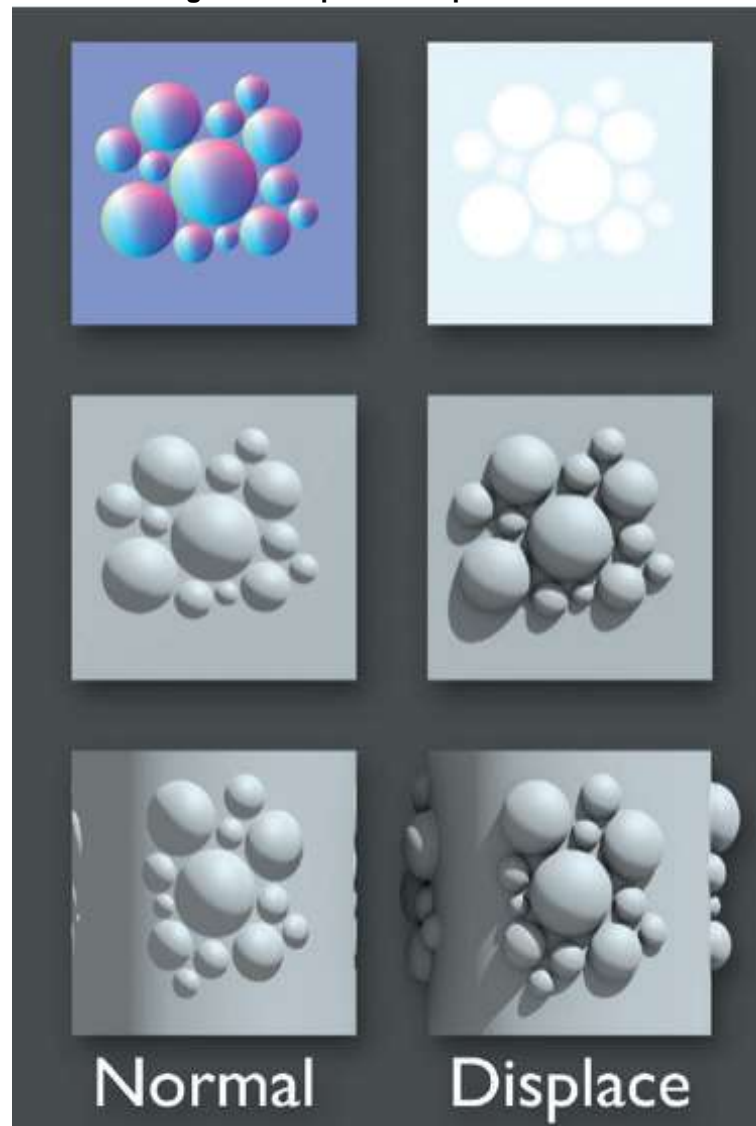
De estos, Tangent es probablemente el más importante; le permite hornear las normales de una malla a la superficie de otra habilitando la configuración Seleccionado a Activo, seleccionando su malla de origen (o mallas) y el objetivo, y luego horneando. Esto representa las normales de la malla de origen en relación con la malla de destino seleccionada. Este mapa se puede usar para distorsionar las normales de la superficie de la malla cuando se renderiza, dando la impresión (cuando el mapa se aplica como parte del material del objeto) de tener muchos detalles en la superficie cuando de hecho la geometría subyacente puede ser mucho más simple. Los mapas normales de espacio tangente son particularmente útiles

porque las normales se registran en relación con la superficie de la malla, por lo que funcionan incluso cuando la malla está distorsionada de su forma original por los modificadores o edición adicional, siempre y cuando las coordenadas UV sigan siendo las mismas.

Texturas Esto hornea el color difuso de cualquier material y color aplicado a la malla. Esto puede ser útil para hornear colores y texturas de procedimiento en un mapa UV.

Desplazamiento Convierte la distancia entre dos mallas en una imagen en blanco y negro. El negro representa el desplazamiento negativo, el gris medio representa el desplazamiento cero y el blanco representa el desplazamiento positivo. Use la opción Seleccionado para activo para hornear desde una malla de origen (o mallas) a la malla activa. El mapa de desplazamiento generalmente se usa como una alternativa a los mapas normales, y este mapa se puede usar como entrada para el modificador de desplazamiento o la configuración de desplazamiento de un material para deformar una malla en la forma de la malla de origen. Alternativamente, un mapa de desplazamiento horneado puede usarse como un mapa de relieve para un material. (Los mapas de relieve funcionan como mapas normales, dando la impresión de detalles de la superficie al alterar el sombreado de la superficie, pero solo requieren entrada en blanco y negro). Los mapas normales son una forma muy eficiente de dar la impresión de detalles, pero no afecta la silueta de la malla o la proyección de sombras porque no se desplaza ninguna geometría real. El mapeo de desplazamiento crea detalles reales al deformar la malla, pero requiere que subdivida la malla para proporcionar la geometría suficiente para deformarse, un grupo de esferas se ha horneado en un mapa normal (8 bits) y un mapa de desplazamiento (32 bits). Estos mapas se han aplicado a un plano y un cilindro. Los objetos mapeados normales capturan gran parte del sombreado con polycounts mucho más bajos, pero no afectan la forma real de la malla o las sombras proyectadas. Los objetos mapeados por desplazamiento son mucho más realistas, pero deben subdividirse para capturar todos los detalles. (Simons, 2013, pp: 141)

Figura 33: Tipos de mapa de textura



Fuente: Simons, 2013

Alfa: Esto hace que la transparencia alfa de la malla se convierta en una textura. Es útil para hornear valores de procedimiento en una textura de imagen.

Emisión: Esto hornea el color de emisión y la cantidad de un material en una textura. Al igual que con el mapa de textura, esto puede ser útil para hornear colores de emisión de procedimiento en una textura de imagen.

Colores de espejo: Esto hornea el color de los reflejos trazados por rayos para un material en una textura. Es útil para hornear valores de procedimiento en una textura de imagen.

Intensidad de espejo: Esto hornea la cantidad de reflexión de trazado de rayos para un material en una textura y es útil para hornear valores de procedimiento para una textura de imagen.

Colores especulares: Esto hornea el color de los reflejos especulares de un material en una textura y es útil para hornear valores de procedimiento para una textura de imagen.

Intensidad especular: Esto hornea la cantidad de reflejo especular de un material en una textura. Es útil para hornear valores de procedimiento en una textura de imagen.

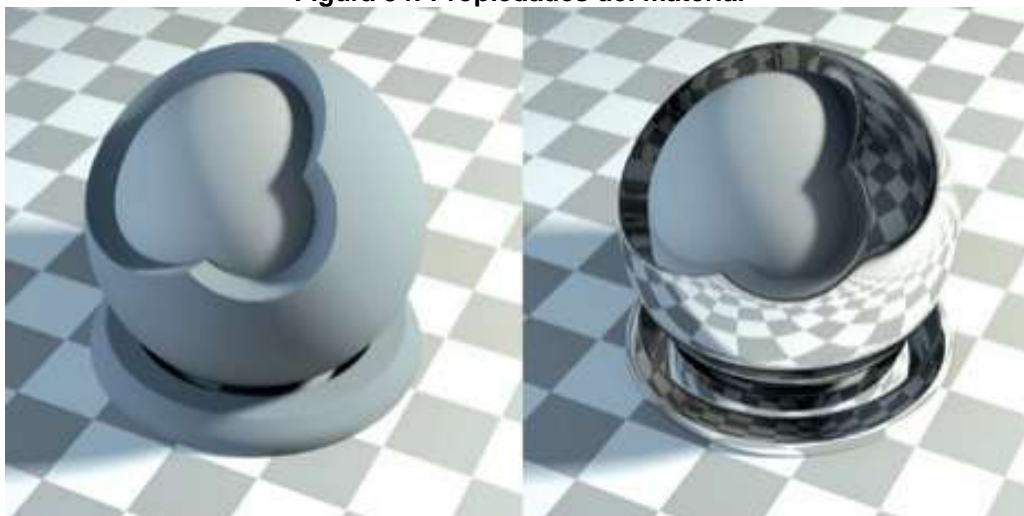
Cada tipo de mapa de textura tiene un propósito de vez en cuando. De los mapas de textura, estos mapas son los más útiles para usar con texturas pintadas y para representar modelos finales detallados. (Simons, 2013, pp. 140-141)

2.4.3. Materiales.

El motor de renderización es cómo Blender crea una imagen final utilizando toda la información de su escena: objetos, materiales, luces y configuraciones. Blender tiene dos motores de render. El más antiguo, el motor de renderizado interno de Blender, es un renderizado muy robusto (aunque un poco anticuado) que utiliza varios trucos y técnicas para crear el render final, evitando la precisión física absoluta para la velocidad y la flexibilidad. Sin embargo, esta falta de precisión física no significa que Blender Internal no pueda generar imágenes de aspecto realista. Puede renderizar escenas rápidamente con una variedad de materiales complejos para crear una apariencia final en cualquier lugar entre una estética altamente realista y una muy poco realista y estilizada. Esto hace que Blender Interno sea un renderizador altamente flexible, muy adecuado para la animación. Cycles es el nuevo motor de render de Blender, que todavía está en desarrollo activo. A diferencia del motor interno, se enfoca en crear simulaciones de luz más realistas y utiliza modelos más físicamente correctos para materiales y luces. Como resultado, puede ofrecer imágenes muy realistas, incluidos los efectos que son difíciles de lograr en Blender Internal, como la refracción compleja de la luz en el vidrio y otros objetos transparentes, múltiples reflejos de la luz que rebota alrededor de una escena y la

emisión físicamente correcta. de luz de los objetos. La compensación es que (al menos a partir de este escrito) Cycles renderiza escenas, incluso aquellas con iluminación simple, más lentamente que Blender Internal. Y debido a que Cycles está en desarrollo activo, carece de ciertas características importantes, como la dispersión subsuperficial y la capacidad de representar el cabello y la piel. El renderizador que se eligió para cada proyecto tiene un impacto significativo en cómo defino mis materiales, porque estos dos motores de render funcionan de diferentes maneras y requieren que los materiales se configuren de manera diferente. Para Bat Creature, usaré Blender Internal para renderizar una piel realista con dispersión subsuperficial, así como el pelaje. Y para el Spider Bot, aunque cualquier renderizador lo haría, usaré Cycles para crear algunos materiales brillantes y reflectantes. Para el proyecto Jungle Temple, usaré Cycles para permitirme experimentar con la iluminación y obtener retroalimentación directa, gracias a la vista previa interactiva de renderizado de Cycles. Antes de examinar las diferencias en la creación de materiales para los dos motores de render diferentes, echemos un vistazo a algunos principios generales que surgirán a medida que discutamos los materiales. Estos principios se centran principalmente en las diversas formas en que una superficie puede interactuar con la luz al reflejarla, refractarla o absorberla.

Figura 34: Propiedades del material

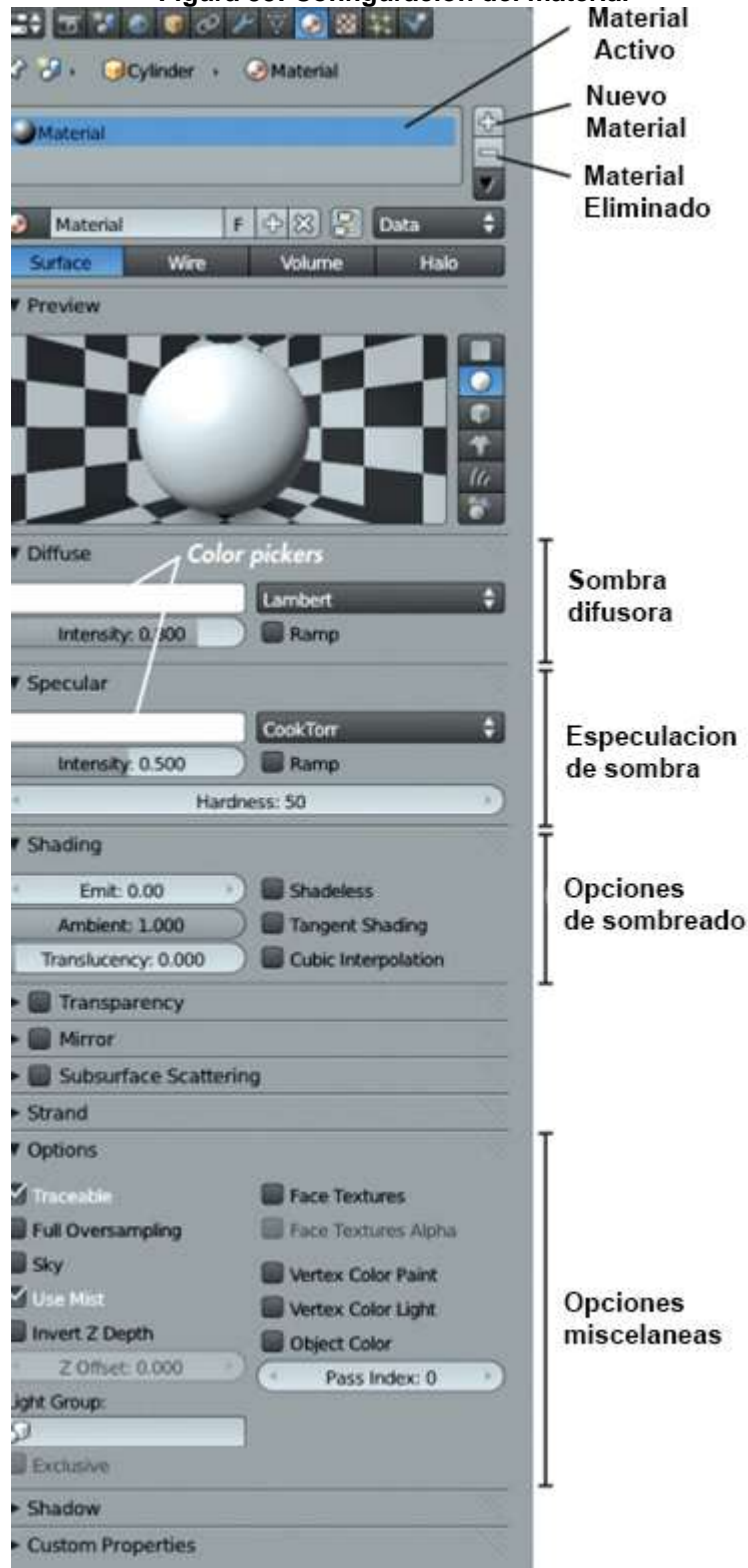


Fuente: Simons, 2013

Blender Internal es el motor de renderización predeterminado de Blender, y puede crear materiales para Blender Internal utilizando las pestañas Materiales y Texturas

del editor de Propiedades. Puede encontrar las propiedades de los materiales en la pestaña Material (y las texturas y aspectos de un material que pueden usarse para afectar) en la pestaña Texturas. Los materiales de Blender Internal son bastante modulares, lo que le permite agregar y combinar diferentes propiedades del material, tales como reflexiones especulares, reflectividad de espejo, transparencia y dispersión subsuperficial, simplemente activando las opciones relevantes de los diferentes paneles en la Figura 35 del material: Ray tracing vs. Z Transparency (usando el renderizador interno de Blender). Izquierda: Transparencia Z con un alfa de 0.25 y reflejos especulares. La visibilidad difusa especular y leve hace que este material sea visible. Derecha: transparencia de trazado de rayos con un IOR de 1.5. Esta imagen es más realista, pero tardará mucho más en renderizarse. Pestaña, sin tener que comenzar desde cero. Observe en la figura que todos los materiales asignados al objeto activo están en el panel en la parte superior de la pestaña. A continuación, en el panel Vista previa, hay una vista previa del material activo, seguido de otros paneles que contienen la configuración de Materiales. Discutiré los paneles relevantes en la pestaña Materiales. (Simons, 2013, pp: 184-185)

Figura 35: Configuración del material



Fuente: Simons, 2013

Difuso: Este panel contiene la configuración para el color difuso y el sombreado de un material. El selector de color le permite establecer el color difuso del material, que luego se multiplica por el valor de intensidad. El menú desplegable a la derecha le permite configurar el modelo de sombreador difuso, que afecta la caída del color difuso del material de claro a oscuro en diferentes ángulos. El modelo de sombreador Lambert predeterminado suele estar bien, aunque el modelo de sombreador Oren Nayar se puede utilizar para imitar superficies con una estructura microscópica más rugosa, como arcilla o piedra. Los otros modelos de sombreadores aquí son más exóticos y raramente útiles.

Especular: Los ajustes en el panel Especular son similares a los del panel Difuso, excepto que controlan el sombreado especular de un objeto (una aproximación) aproximada de la reflexión especular. La entrada de dureza determina cuán amplias o apretadas son estas reflexiones. Los materiales con un alto valor de dureza tienen reflejos nítidos y brillantes, mientras que aquellos con un valor de dureza bajo tienen reflejos más suaves y extendidos.

2.4.3.1. Sombreado.

Las siguientes opciones afectan el sombreado general de un material:

Emitir: Esto hace que un objeto parezca brillante incluso cuando no está iluminado e incluso puede hacer que un objeto arroje luz sobre otros objetos cuando se usa con ciertos ajustes del Mundo. Ambient Esto determina si un objeto recibe iluminación ambiental (usando la configuración de Color ambiental en la pestaña Configuración mundial).

Translucidez: Esto hace que un objeto aparezca iluminado tanto en su frente como en su parte posterior, haciendo que se vea translúcido, como papel delgado o una hoja.

Sin sombra: Esto elimina completamente el efecto de la luz en un objeto; el objeto obtendrá su color y brillo por completo de su color difuso y texturas.

Sombreado tangente: Altera el sombreado de una superficie para que se parezca más al metal cepillado u otros materiales con un "grano" o direccionalidad a su

estructura microscópica. La dirección del grano se toma de las coordenadas UV del objeto, por lo que el grano se alinearán con el eje v en el espacio UV. En otras palabras, se alinearán verticalmente en el editor de imágenes UV, así que asegúrese de haber desenvuelto el objeto adecuadamente al habilitar esta opción.

Interpolación cúbica: Esto altera la forma en que la superficie se mezcla entre la luz y la sombra, generalmente con resultados más suaves a menudo un poco más oscuro.

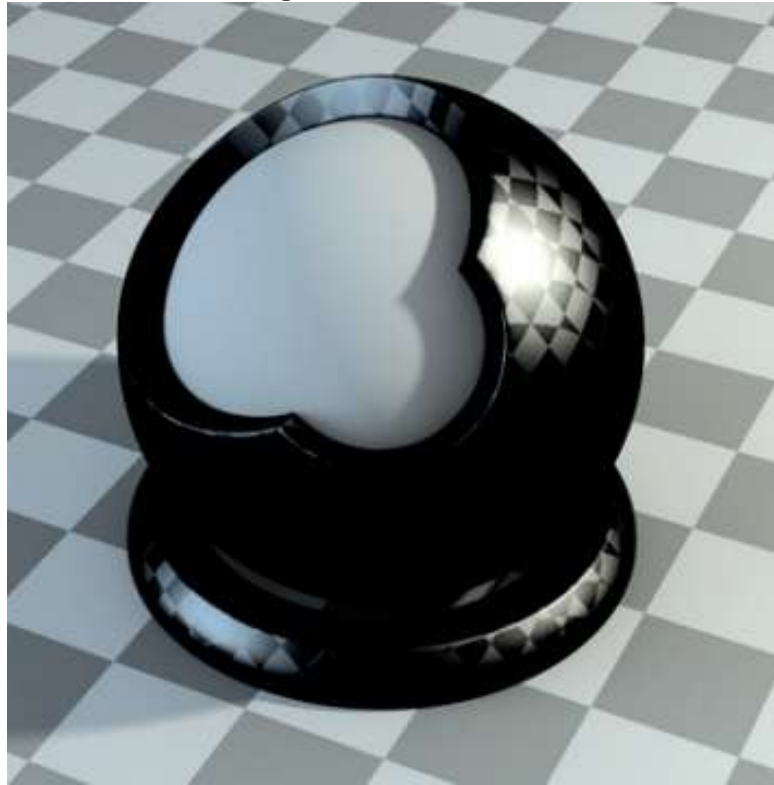
Transparencia Active: la casilla de verificación Transparencia para representar un material con transparencia. Este panel ofrece tres formas de representar objetos transparentes, utilizando el valor alfa de la superficie. La más simple, Máscara, simplemente combina el material en el fondo donde el color es transparente. La transparencia es un poco más sofisticada, representando lo que esté detrás del objeto transparente. La transparencia con trazado de rayos calcula la refracción adecuada. Al elegir Transparencia con trazado de rayos, aparecen varias opciones que le permiten establecer el índice de refracción del material y el brillo y la cantidad de filtrado realizado por el material.

Espejo Habilite: Espejo para activar la representación de reflejos con trazado de rayos. La configuración de este panel le permite definir la cantidad (Reflectividad) y la nitidez (Cantidad brillante) de reflejos trazados por rayos.

Dispersión del subsuelo (SSS): Esta opción controla el efecto de la dispersión de la luz debajo de la superficie de un material que se encuentra comúnmente en la piel o la cera. Las opciones de este panel contienen algunas opciones diversas. Por ejemplo, la configuración Rastreable controla si un objeto se tiene en cuenta al representar sombras y reflejos con trazado de rayos.

Sombra: Este panel controla cómo un objeto recibe y proyecta sombras. Por ejemplo, la opción Lanzar solo hace que un objeto parezca invisible, excepto por las sombras que proyecta. Shadows Only representa las sombras de un material como las únicas partes no transparentes. (Esto es particularmente bueno para crear sombras que luego puedes componer en otra imagen.

Figura 36: Sombreado



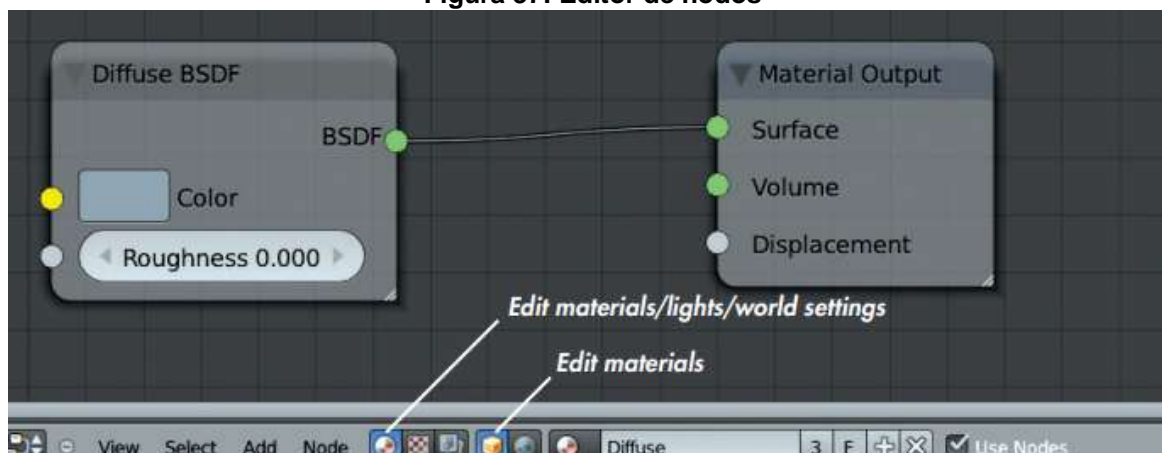
Fuente: Simons, 2013

2.4.3.2. Ciclos materiales.

A diferencia del renderizador interno de Blender, que está repleto de trucos sucios y aproximaciones para agilizar el renderizado, el renderizador de Cycles utiliza modelos más realistas físicamente para los materiales. Combina realismo con un enfoque basado en nodos que se centra en la construcción de materiales complejos a partir de piezas simples. Todavía puede usar la pestaña Materiales del editor de Propiedades para editar los materiales de Cycles, que ofrece una visión general del material actual, pero es mucho más simple y claro usar el editor de Nodos de Blender, como se muestra en la Figura 38. Para crear un material con Ciclos, todo lo que necesita es un nodo Sombreador y un nodo Salida de material. Puede construir desde allí y combinar otros sombreadores, entradas y texturas para crear una amplia variedad de materiales. Para agregar un nodo, use shift-A y elija el nodo que desea del menú que aparece, al igual que agregar objetos en la ventana 3D. Los nodos Shader BSDF (función de distribución de dispersión bidireccional) controlan el

funcionamiento de los materiales. Estas funciones definen cómo interactúa la luz con una superficie, por ejemplo, si el material refleja la luz de manera difusa, actúa como un espejo o transmite luz como el vidrio. Encontrará los sombreadores BSDF en Sombreadores en el menú Agregar en el encabezado, junto con algunas otras opciones. (Simons, 2013, pp: 187-188)

Figura 37: Editor de nodos



Fuente: Simons, 2013

Esto afecta el fondo del mundo en lugar de los materiales.

BSDF difuso: Dispersa la luz en todas las direcciones, creando un aspecto difuso.

BSDF brillante: Esto refleja la luz como un espejo. Al aumentar el valor de rugosidad, los reflejos se vuelven cada vez más borrosos.

Glass BSDF: La función Glass BSDF transmite la luz como el vidrio o cualquier otro medio transparente. El índice de refracción (IOR) determina cuánto se dobla la luz cuando penetra en una superficie. Cuanto mayor sea el valor, más se dobla. Los materiales más densos tienen IOR más altos. Por ejemplo, el IOR del vidrio es de alrededor de 1.5, y el IOR del agua es de aproximadamente 1.3. El diamante, un material mucho más denso, tiene un IOR de alrededor de 2.4.

BSDF translúcido: Esto dispersa la luz en todas las direcciones desde la parte posterior del objeto, dando como resultado una apariencia translúcida, como la de un papel u hoja delgada, con luz transmitida a través del material.

BSDF transparente: Esto permite que la luz penetre en un material sin ser refractada, como si no hubiera nada allí. Es útil cuando se combina con mapas alfa.

Velvet BSDF: Esto refleja la luz algo así como el sombreador difuso, pero refleja más la luz cuando se ve desde ángulos oblicuos. El resultado es una apariencia aterciopelada.

Emisión: Esto convierte un objeto en una lámpara que emite su propia luz. Retención Esto crea un "agujero" en una imagen que pasa al color de fondo con transparencia alfa cero. Es útil para componer.

Mix Shader: Esto le permite mezclar dos shaders con la proporción de mezcla determinada por la entrada del factor. Los nodos Mix le permiten combinar sombreadores en numerosas combinaciones para crear todo tipo de materiales. Por ejemplo, podría crear un material similar al plástico combinando un sombreador BSDF brillante y BSDF difuso,

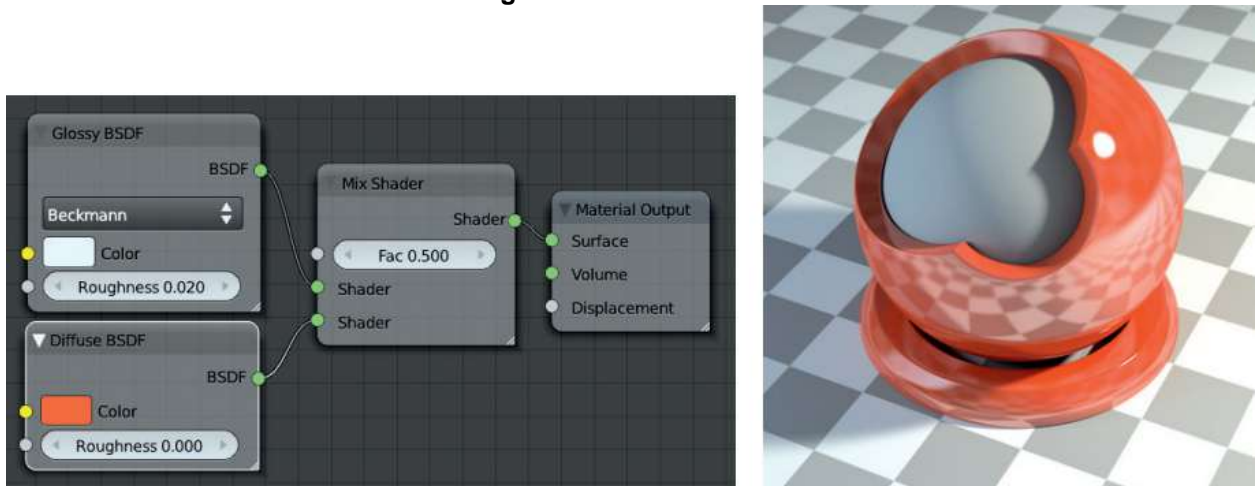
Agregar sombreador: Esto agrega dos sombreadores juntos. Es menos flexible que el nodo Mix porque no puede controlar la cantidad de mezcla, pero funciona de la misma manera.

Figura 38: Glass BSDF



Fuente: Simons, 2013

Figura 39: Mix shader



Fuente: Simons, 2013

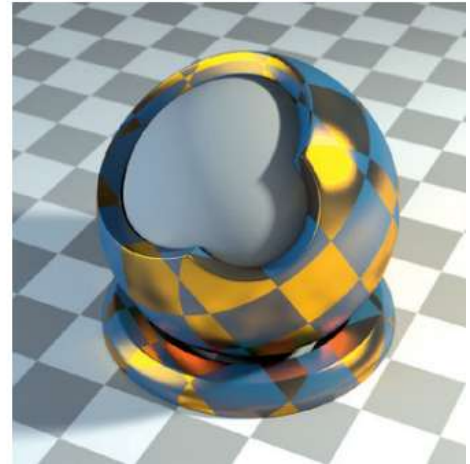
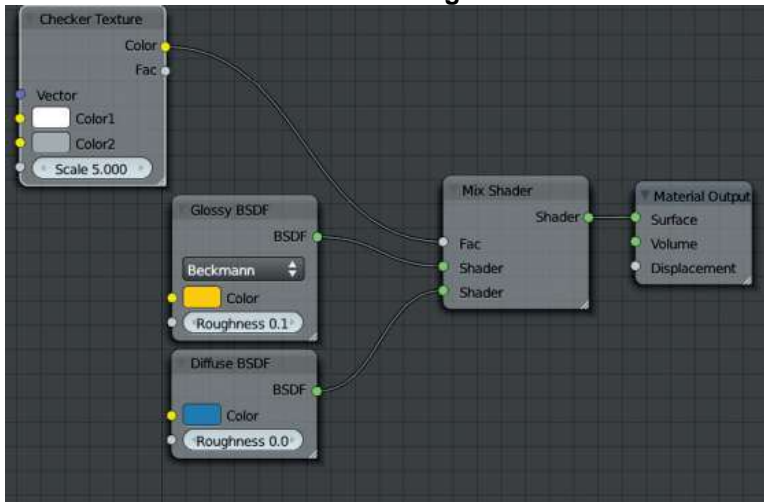
2.4.3.3. Nodos de textura.

Los nodos Texture son particularmente útiles para crear materiales, y pueden proporcionar una variedad de texturas basadas en imágenes y procedimientos para usar en sus materiales. Los otros nodos crean texturas de procedimiento. Por ejemplo, la textura de ruido crea nubes multicolores que son útiles para agregar detalles de aspecto aleatorio a las superficies, y la textura de las ondas crea bandas repetitivas que pueden distorsionarse para parecerse a un grano de madera. El tablero de ajedrez es una gran textura para probar la configuración del material; por ejemplo, al conectar un nodo de tablero de ajedrez a la entrada Fac de un nodo Mix Shader, puede usarlo para comparar dos shaders. (Simons, 2013, pp: 189)

2.4.3.4. Nodos en Blender Internal.

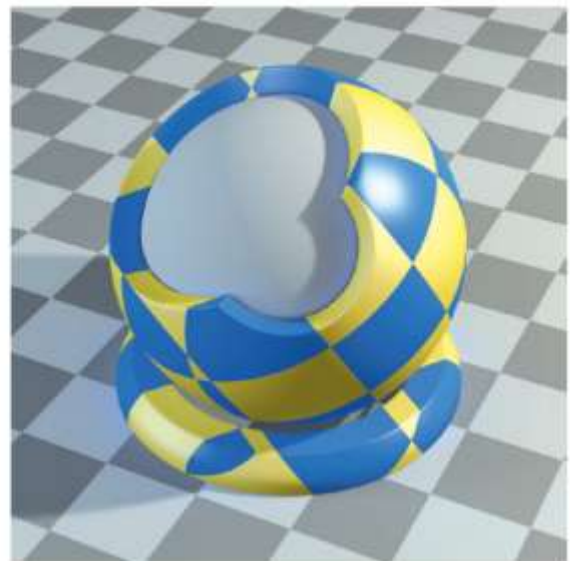
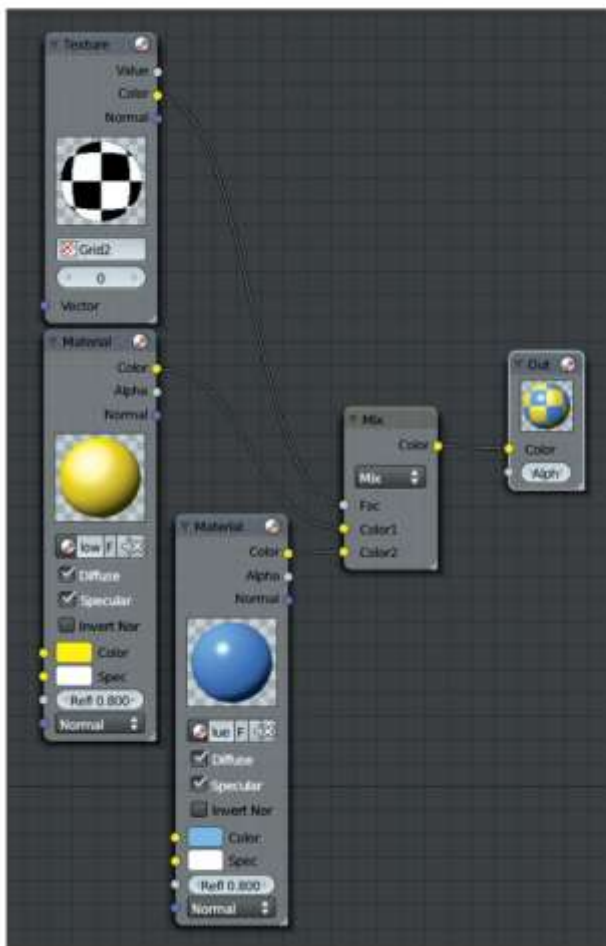
Si bien no los cubriré en detalle, Blender Internal también admite materiales de nodo. Aunque los principios son los mismos, estos funcionan de forma ligeramente diferente a los nodos de Cycles. En lugar de sombreadores, puede usar otros materiales como entradas y combinarlos para crear nuevos materiales. (Simons, 2013, pp: 189)

Figura 40: Texturas con ciclos



Fuente: Simons, 2013

Figura 41: Combinación de materiales



Fuente: Simons, 2013

2.5. FÍSICA.

La física es más que una rama de las ciencias físicas: es la más fundamental de las ciencias. Estudia la naturaleza de realidades básicas como el movimiento, las fuerzas, energía, materia, calor, sonido, luz y el interior de los átomos. La química estudia la manera en que está integrada la materia, la manera en que los átomos se combinan para formar moléculas y la manera en que las moléculas se combinan para formar los diversos tipos de materia que nos rodea. La biología es aún más compleja, pues trata de la materia viva. Así, tras la biología esta la química y tras la química esta la física. (Bragado, 2018, pp: 14)

2.5.1. Cinemática.

Cinemática es la parte de la física que estudia el movimiento de los cuerpos, aunque sin interesarse por las causas que originan dicho movimiento. Un estudio de las causas que lo originan es lo que se conoce como dinámica. Las magnitudes que define la cinemática son principalmente tres, la posición, la velocidad y la aceleración. (Bragado, 2003, pp: 29)

Posición es el lugar en que se encuentra el móvil en un cierto instante de tiempo t . Suele representarse con el vector de posición \vec{r} . Dada la dependencia de este vector con el tiempo, es decir, si nos dan $\vec{r}(t)$, tenemos toda la información necesaria para los cálculos cinemáticos.

Velocidad es la variación de la posición con el tiempo. Nos indica si el móvil se mueve, es decir, si varía su posición a medida que varía el tiempo. La velocidad en física se corresponde al concepto intuitivo y cotidiano de velocidad.

Aceleración indica cuanto varía la velocidad al ir pasando el tiempo. El concepto de aceleración no es tan claro como el de velocidad, ya que la intervención de un criterio de signos puede hacer que interpretemos erróneamente cuando un cuerpo se acelera ($a > 0$) o cuando se “decelera” ($a < 0$). Por ejemplo, cuando lanzamos una piedra al aire y ésta cae es fácil ver que, según sube la piedra, su aceleración es negativa, pero no es tan sencillo constatar que cuando cae su aceleración sigue

siendo negativa porque realmente su velocidad está disminuyendo, ya que hemos de considerar también el signo de esta velocidad.

2.5.1.1. Tiro Parabólico.

Se denomina tiro parabólico, en general, a aquellos movimientos que suceden de forma bidimensional sobre la superficie de la tierra. Para este tipo de móviles el movimiento se descompone en sus componentes x e y. El movimiento en x no sufre aceleración, y por tanto sus ecuaciones serán:

$$\text{Eje x} \begin{cases} x &= x_0 + v_{0x}t \\ v_x &= v_{0x} \end{cases} \quad \text{Ec - 5}$$

pero en cambio en el eje y se deja sentir la fuerza de la gravedad, supuesta constante y por tanto sus ecuaciones serán.

$$\text{Eje y} \begin{cases} y &= y_0 + v_{0y}t - \frac{1}{2}gt^2 \\ v_y &= v_{0y} - gt \end{cases} \quad \text{Ec - 6}$$

Donde:

x_0 : Coordenadas x en instancia 0.

y_0 : Coordenadas y en instancia 0.

t : Tiempo.

g : Gravedad.

Algunas preguntas típicas del tiro parabólico son calcular el alcance y altura máxima. Estas preguntas se pueden contestar sabiendo que la altura máxima se alcanzara cuando $V(y) = 0$. De esta condición se extrae el tiempo que tarda en alcanzar la altura máxima y sustituyendo en la ecuación de las y se obtiene la altura máxima. El alcance máximo se puede calcular razonando que, para cuando esto suceda, el móvil volver a estar al nivel del suelo y por tanto $y = 0$, sustituyendo se obtiene t y,

sustituyendo éste en las x el resultado. Otras cantidades se pueden conseguir de manera similar.

- x_0 e y_0 serán las coordenadas donde el móvil se encuentra en el instante $t = 0$, inicio del movimiento, y v_{x0} y v_{y0} la velocidad con la que se mueve en ese instante. Si nos han indicado que el móvil se movía con una velocidad v formando un ángulo α con la horizontal se puede ver muy fácilmente que, entonces, $v_{x0} = v \cos \alpha$ y $v_{y0} = v \sin \alpha$. A su vez el significado de las variables x e y es el siguiente: éstas nos indican a qué distancia horizontal (x) y altura (y) se encuentra el móvil en cada instante de tiempo t , considerando que estamos tomando como origen para medir estas distancias horizontales y alturas desde el sistema de coordenadas respecto al cual estemos tomando todos los demás datos.
- Se podría hacer un estudio más complejo incluyendo el rozamiento del aire. Para esto habrá que modificar las ecuaciones x e y a las nuevas ecuaciones deducidas en el apéndice B.

2.5.2. Ley de la inercia.

La ley de la inercia se podría enunciar como:

- Todo cuerpo permanece en su estado actual de movimiento con velocidad uniforme o de reposo a menos que sobre él actúe una fuerza externa neta o no equilibrada.

donde la fuerza neta de la que hablamos antes sería la suma vectorial de todas las fuerzas que puedan actuar separadamente sobre el cuerpo.

- Esta es la razón por la cual es tan peligroso para los astronautas en el espacio separarse de la nave sin un cordón que los una a ella, ya que, si chocan con algo y salen impulsados, como no actúa ninguna fuerza sobre ellos, seguirán desplazándose uniformemente y separándose de la nave sin posibilidad de volver a ella (a no ser que tengan un pequeño impulsor). (Bragado, 2003, pp: 34)

2.5.3. Segunda ley de Newton.

Esta ley es la más importante en cuanto nos permite establecer una relación numérica entre las magnitudes “fuerza” y “aceleración”. Se podría enunciar como. La aceleración que toma un cuerpo es proporcional a la fuerza neta Recuerda externa que se le aplica. La constante de proporcionalidad es la masa del cuerpo, con lo que numéricamente esta expresión se denota como.

$$F = m * a \quad \text{Ec - 7}$$

Donde:

F : Fuerza.

m : Masa.

a : Aceleración.

Esta expresión nos relaciona F , m y a de una forma unívoca. Básicamente nos dice que el resultado que producen una serie de fuerzas sobre un cuerpo es que dicho cuerpo se acelere en la misma dirección y sentido que la suma de las fuerzas que le son aplicadas y con una intensidad o modulo que será la misma que la resultante de las fuerzas dividida entre la masa del cuerpo. Así pues, un cuerpo experimenta una aceleración mientras está siendo sometido a una fuerza resultante no nula. Si dicha fuerza cesa el cuerpo adquiriría un movimiento rectilíneo uniforme o se quedaría quieto, según el caso. (Bragado, 2003, pp: 35)

2.5.4. Tercera ley de Newton.

La tercera ley de Newton expresa una interesante propiedad de las fuerzas: éstas siempre se van a presentar en parejas. Se puede enunciar como Recuerda. Si un cuerpo A ejerce, por la causa que sea, una fuerza F sobre otro B, este otro cuerpo B ejercerá sobre A una fuerza igual en modulo y dirección, pero de sentido contrario. Gracias a esta ley¹ se pueden entender fenómenos como que, para saltar hacia arriba ¡empujamos la Tierra con todas nuestras fuerzas hacia abajo! Al hacer esto la Tierra también ejerce esta misma fuerza con nosotros, pero con sentido contrario (es decir, hacia arriba) y como la masa de la Tierra es enorme en comparación con la nuestra, el resultado es que nosotros salimos despedidos hacia arriba, pero la Tierra

no se mueve apreciablemente. Así también si empujamos una superficie puntiaguda con mucha fuerza, podemos clavárnosla, porque dicha superficie también estará empujando nuestro dedo con la misma fuerza que nosotros a ella, y como la superficie de la aguja es muchísimo menor la presión que esta hace sobre nuestro dedo es muy grande. Nota ! Entonces, si a toda fuerza que se ejerce se opone otra de sentido contrario ¿no deberían anularse las fuerzas y nada se podría mover? No, porque las fuerzas se ejercen en cuerpos diferentes. Así en el ejemplo del salto, nosotros empujamos a la Tierra y la Tierra a nosotros, pero estas fuerzas no se anulan porque, como es evidente, nosotros y la Tierra somos cuerpos distintos. (Bragado, 2003, pp: 36)

2.5.4.1. Momento Lineal.

La ley de Newton, expresada como $F = m a$ puede ser utilizada también para demostrar otras relaciones interesantes, siempre que se manipule adecuadamente. Por ejemplo, si definimos una cantidad ($p \rightarrow a$) la que llamaremos cantidad de movimiento, podemos decir que una fuerza es la encargada de variar la cantidad de movimiento sobre un cuerpo. De esta forma definamos p tal que:

$$F = \frac{d}{dt} * p \quad \text{Ec - 8}$$

Donde:

dt : Diferencial con respecto del tiempo.

p : cantidad (cte).

Una forma intuitiva de comprender el momento lineal es como una forma de medir la dificultad de llevar una partícula hasta el reposo. Así es claro que, cuanto más masivo sea un cuerpo y más velocidad tenga, tanto más nos costaría “parar” el movimiento de dicho cuerpo. (Bragado, 2003, pp: 38)

2.5.4.2. Conservación del momento lineal.

Cuando la resultante de las fuerzas externas sobre un sistema es nula, ¿qué sucede con p ? Como la fuerza es la derivada del momento lineal respecto al tiempo, obtenemos que, cuando la fuerza total es cero, esta cantidad que se deriva debe ser

constante y, por tanto, si $F_{\sim} = \sim 0$ esto supone $p_{\sim} = \text{cte}$. Hemos obtenido así que esta magnitud tan interesante, el momento lineal, se conserva, es decir, no varía, cuando no aparecen fuerzas externas sobre un objeto. Por tanto, podemos decir que:

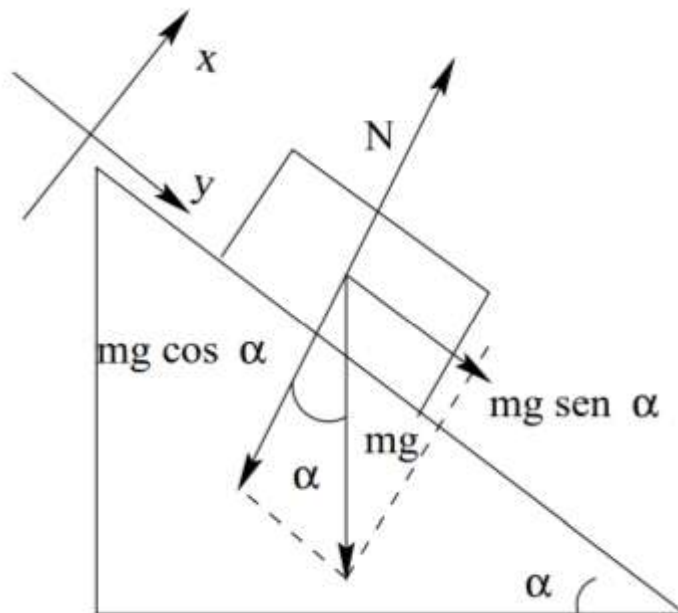
$$p_i = p_f \quad \text{Ec - 9}$$

Donde:

p_i : Constante inicial.

p_f : Constante final.

Figura 42: Descomposición de las fuerzas en un plano inclinado.



Fuente: Bragado, 2003

2.6. INGENIERÍA DE SOFTWARE.

2.6.1. Metodologías Ágiles.

El proceso ágil sigue el ciclo de vida de desarrollo de software que incluye la recopilación de requisitos, análisis, diseño, codificación, prueba y entrega software parcialmente implementado y espera los comentarios de los clientes. (Sheetal Sharma, 2012, pp: 892)

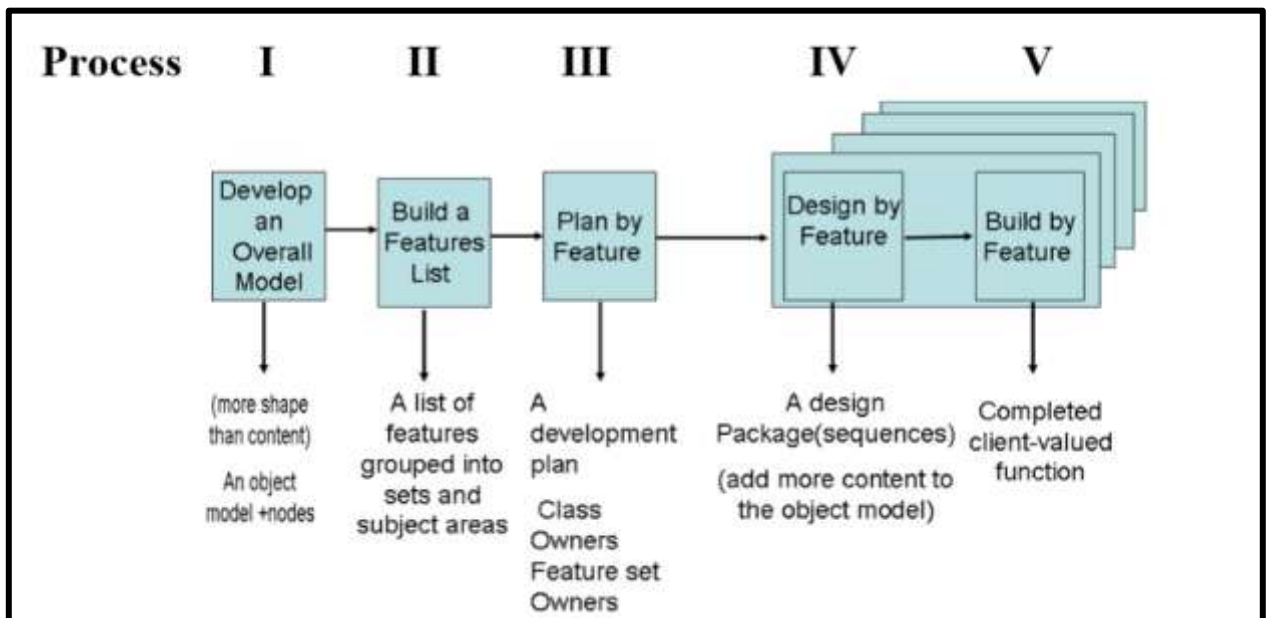
2.6.2. Feature Driven Development (FDD).

FDD es una metodología de desarrollo de software ágil y altamente adaptable que es:

- Alta y corta iterativa.
- Destaca la calidad en todos los pasos
- Ofrece resultados de trabajo frecuentes y tangibles en todos los pasos
- Proporciona información precisa y significativa sobre el progreso y el estado, con el mínimo de
- gastos generales e interrupción para los desarrolladores.
- Es del agrado del cliente, gerentes y desarrolladores

FDD comienza con la creación de un modelo de objeto de dominio en colaboración con Expertos de dominio. Usar información de la actividad de modelado y de cualquier otra actividad requerida que tenga realizado, los desarrolladores crean una lista de características. Luego se elabora un plan general y se asignan responsabilidades. Características de grupos pequeños de características que no duran más de dos semanas para cada grupo y a menudo es mucho más corto se retoman. FDD consta de cinco procesos. (Sadhna Goyal, 2008, pp: 10)

Figura 43: Los cinco procesos de FDD con sus salidas



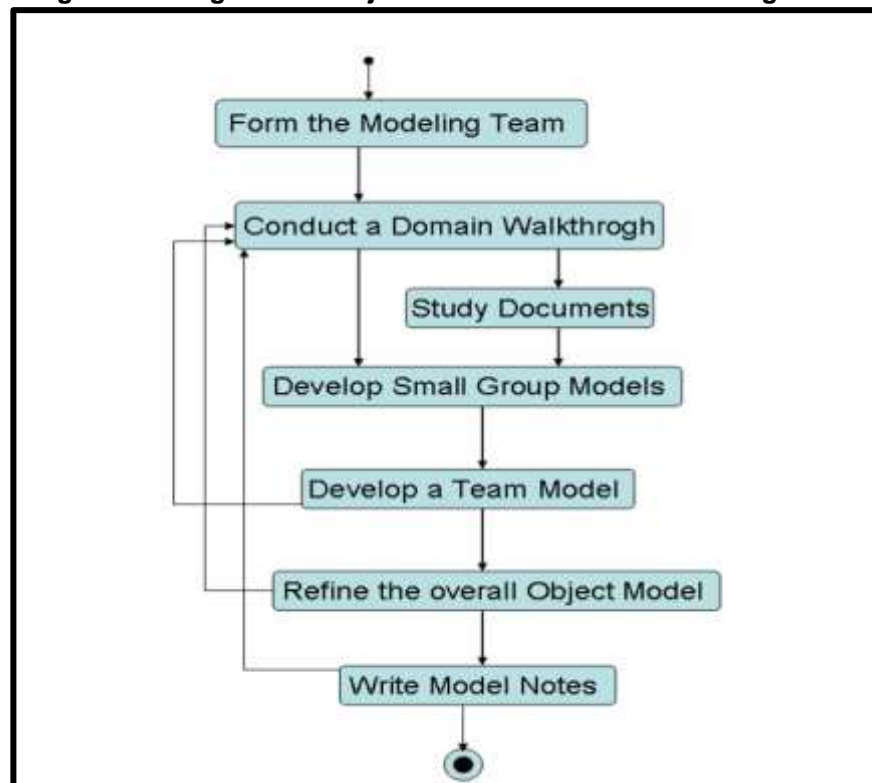
Fuente: Palmer, 2003

2.6.2.1. Desarrollar un modelo general.

Los miembros del equipo de dominio y desarrollo trabajan juntos bajo la guía de un experimentado Modelador de objetos (Arquitecto jefe). Los miembros del dominio

realizan un recorrido inicial de alto nivel del alcance del sistema y su contexto. Luego, los miembros del dominio realizan más detallados tutoriales de cada área del dominio del problema. Después de cada tutorial, el dominio y los miembros de desarrollo trabajan en pequeños grupos para producir modelos de objetos para esa área del dominio. Cada grupo pequeño compone su propio modelo en apoyo del tutorial de dominio y presenta sus resultados para revisión y discusión por pares. Uno de los modelos propuestos o una fusión de modelos se selecciona por consenso y se convierte en el modelo para esa área de dominio. El área de dominio el modelo se fusiona con el modelo general, ajustando la forma del modelo según sea necesario. El modelo de objeto luego se actualiza iterativamente con contenido por el proceso IV. (Sadhna Goyal, 2008, pp. 10)

Figura 44: Diagrama de flujo del desarrollo de un modelo general.

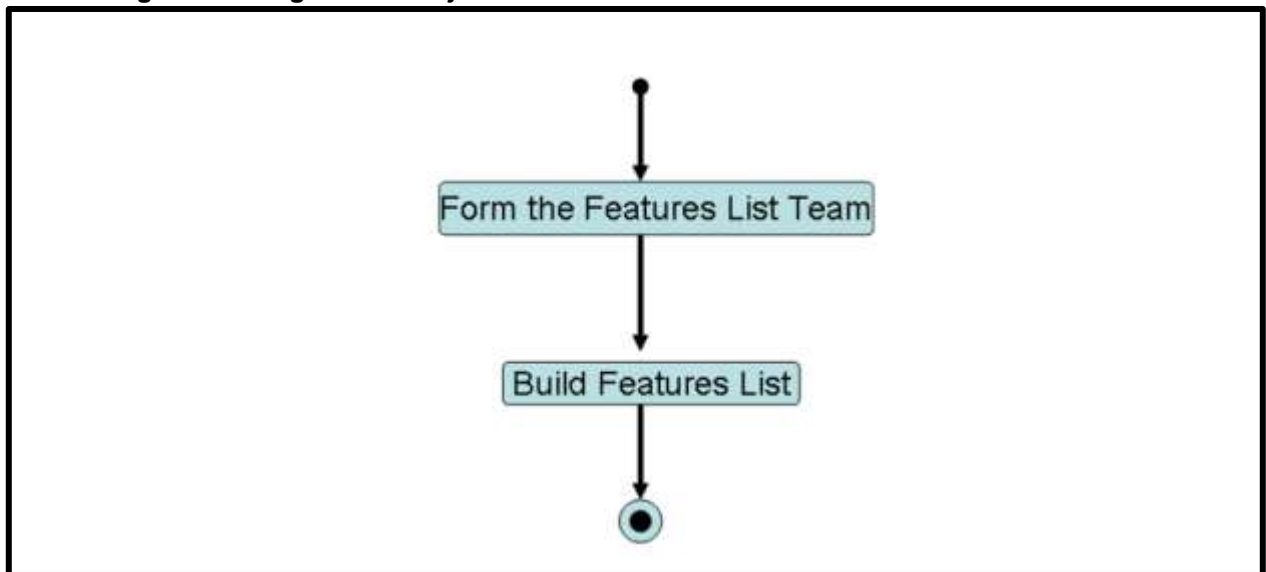


Fuente: Palmer, 2003

2.6.2.2. Crear una lista de características.

Expertos del dominio en proceso 1, el equipo divide el dominio en varias áreas (conjuntos de características principales). Cada área es dividida aún más en una serie de actividades (conjuntos de características). Se identifica cada paso dentro de una actividad. como una característica El resultado es una lista de características categorizadas jerárquicamente. (Sadhna Goyal, 2008, pp. 11)

Figura 45: Diagrama de flujo de la construcción de una lista de características.



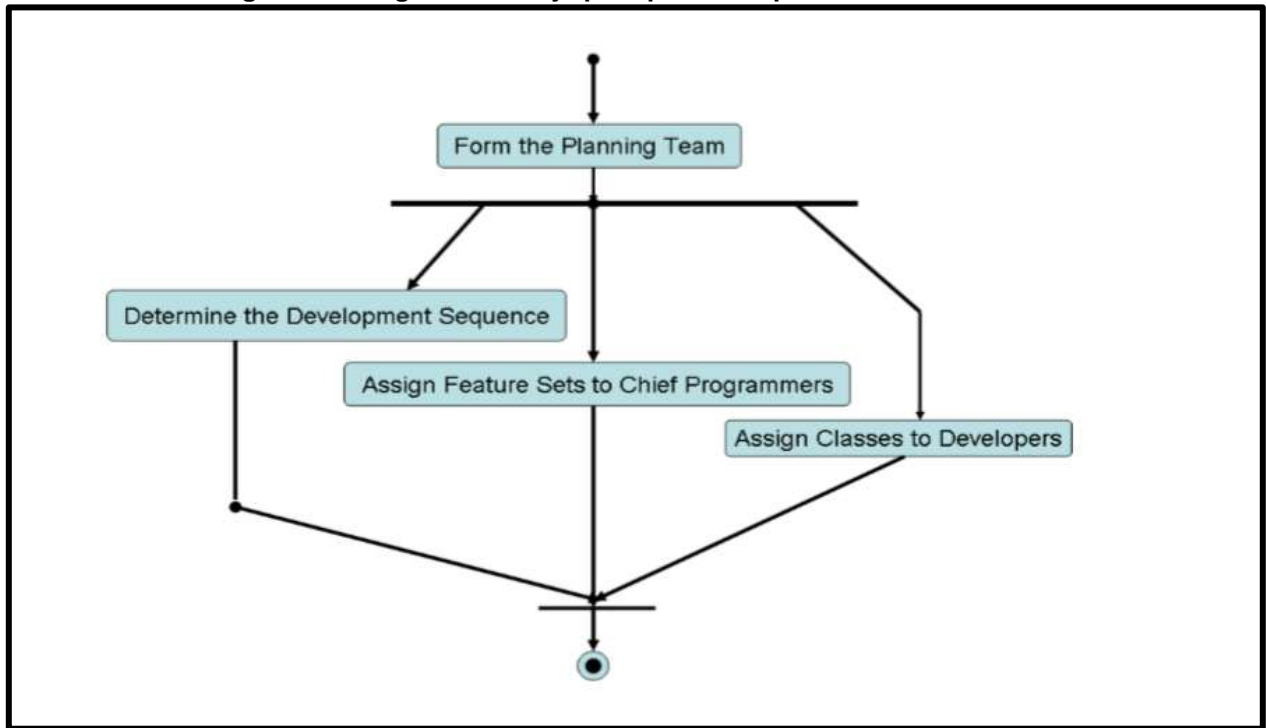
Fuente: Palmer, 2003

2.6.2.3. Plan por característica.

El gerente del proyecto, el gerente de desarrollo y los programadores principales planifican el orden en que las características se implementarán, en función de las dependencias de las características, carga en todo el desarrollo equipo, y la complejidad de las características a implementar. Las tareas principales en este proceso son No es una secuencia estricta. Como muchas actividades de planificación, se consideran juntas, con refinamientos realizados a partir de una o más tareas, luego las otras se consideran nuevamente. Un típico escenario es considerar la secuencia de desarrollo, luego considerar la asignación de conjuntos de características a Los programadores principales, y al hacerlo, consideren cuáles de las clases clave están asignadas a cuáles de Los desarrolladores. Cuando se logra este equilibrio y la secuencia de desarrollo y asignación de Las actividades

comerciales para los programadores principales se completan esencialmente, la propiedad de la clase es terminado. (Sadhna Goyal, 2008, pp: 12)

Figura 46: Diagrama de flujo para planificar por característica.

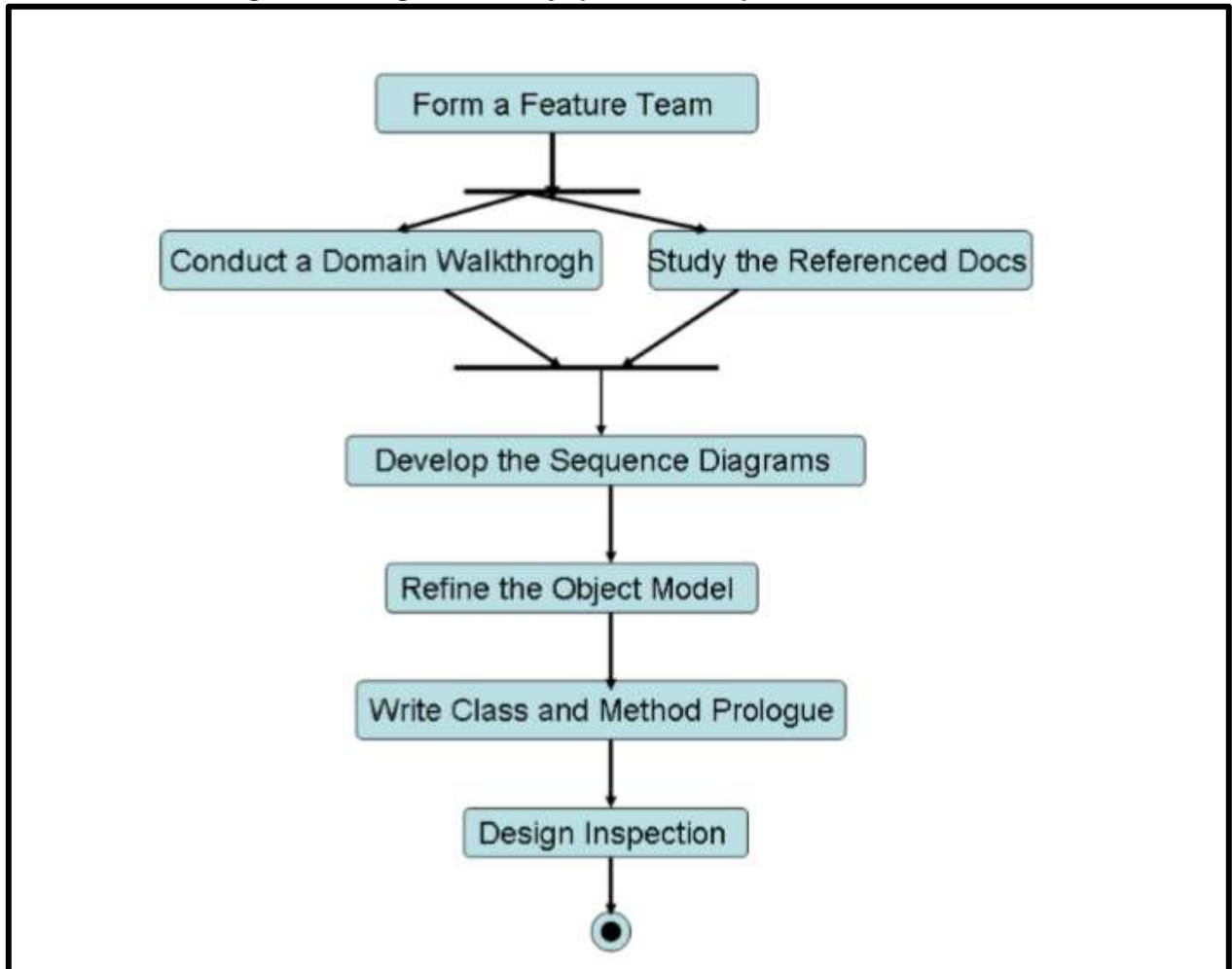


Fuente: Palmer, 2003

2.6.2.4. Diseño por característica.

El desarrollo de una serie de características está programado al asignarlas a un programador jefe. El programador jefe selecciona las características para el desarrollo de su “bandeja de entrada” asignadas características. Operacionalmente, a menudo es el caso que el Programador Jefe programa un pequeño grupo de características a la vez para el desarrollo. Él o ella puede elegir múltiples funciones que utilizan la misma clase (de ahí los desarrolladores). Tal grupo de características forma un trabajo de Programador Jefe Paquete. El programador jefe luego forma un equipo de características identificando a los propietarios de clases (desarrolladores) que probablemente participen en el desarrollo de las características seleccionadas. El jefe Luego, el programador refina el modelo de objeto en función del contenido del diagrama o los diagramas de secuencia. Los desarrolladores escriben prólogos de clase y método. Se lleva a cabo una inspección de diseño. (Sadhna Goyal, 2008, pp. 13)

Figura 47: Diagrama de flujo para diseñar por característica.

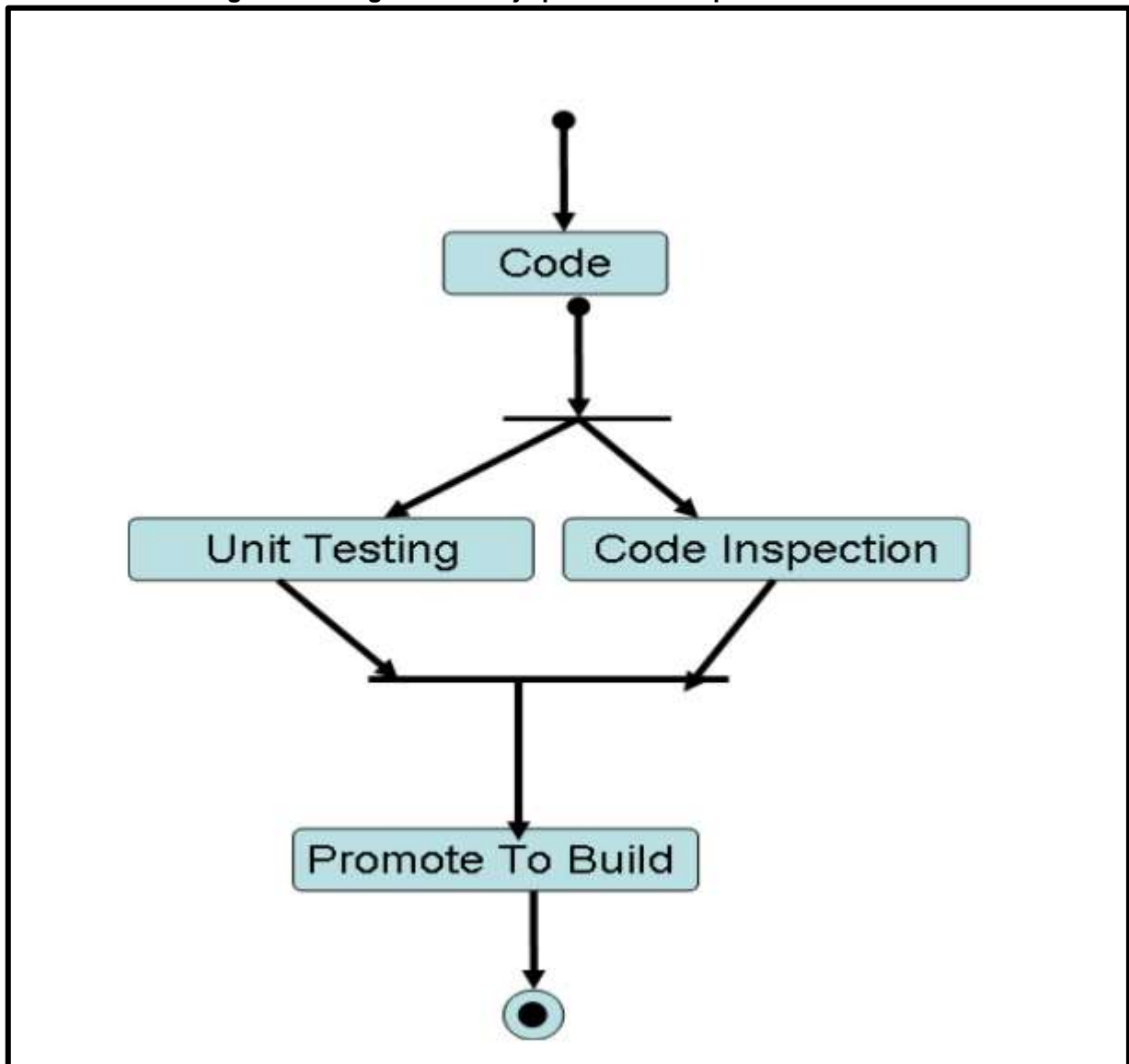


Fuente: Palmer, 2003

2.6.2.5. Construir por característica.

Trabajando desde el paquete de diseño producido durante el proceso construir por característica, la clase los propietarios implementan los elementos necesarios para que su clase admita el diseño de las características en El paquete de trabajo. El código desarrollado es luego probado por la unidad e inspeccionado por código, cuyo orden está determinado por el programador jefe. Después de una inspección exitosa del código, se permite el código para construir. (Sadhna Goyal, 2008, pp: 14)

Figura 48: Diagrama de flujo para construir por característica



Fuente: Palmer, 2003

2.6.3. Arquitecturas de software.

2.6.3.1. Arquitectura de alto nivel (High Level Architecture.)

La arquitectura de alto nivel (HLA) proporciona un marco general dentro del cual los desarrolladores de simulación pueden estructurar y describir sus aplicaciones de simulación. La flexibilidad es el objetivo del HLA. En particular, el HLA aborda dos cuestiones clave: promover la interoperabilidad entre simulaciones y ayudar a la reutilización de modelos en diferentes contextos. Se describen tres componentes principales dentro del conjunto de productos que forman el HLA. El primer

componente, el Marco HLA y la Especificación de Reglas (es decir, este estándar), proporciona un conjunto de diez reglas que juntas aseguran la interacción adecuada de los federados en una federación y definen las responsabilidades de los federados y federaciones. El segundo componente, la plantilla de modelo de objetos (OMT), es una base necesaria para la reutilización y forma un estándar de documentación que describe los datos utilizados por un modelo en particular. El tercer componente, la especificación de la interfaz federada, aborda la interoperabilidad y describe una interfaz de comunicaciones genérica que permite conectar y coordinar modelos de simulación. Aunque el HLA es una arquitectura, no un software, se requiere el uso de software de infraestructura de tiempo de ejecución (RTI) para soportar las operaciones de ejecución de una federación. El software RTI proporciona un conjunto de servicios, según lo definido por la especificación de la interfaz federada, utilizado por los federados para coordinar las operaciones y el intercambio de datos durante una ejecución en tiempo de ejecución.

Las simulaciones son necesariamente abstracciones del mundo real, y ningún diseño de simulación puede satisfacer las necesidades funcionales de toda la comunidad de modelado y simulación. Sin embargo, al definir una arquitectura superior, se pueden abordar problemas genéricos. Al hacerlo, es esencial que dicha arquitectura abarca tanto entornos informáticos diferentes como diferentes clases de simulaciones. Este estándar, que describe el marco y las reglas, tiene la intención de proporcionar algo de la filosofía general detrás del HLA, que incluye orientación sobre cómo diseñar, usar y adherirse a la visión del HLA. (IEEE, 2010, pp: IV)

2.6.3.2. Modelos de objetos.

Hay tres tipos de modelos de objetos en el marco HLA. Primero es un modelo de objeto de simulación (SOM). El SOM es una especificación de los tipos de información que una aplicación de federación individual podría proporcionar a las federaciones de HLA, así como la información que un federado unido individual puede recibir de otros federados unidos en las ejecuciones de federaciones de HLA. El SOM puede verse como la expresión nativa de una aplicación federada de sus capacidades e interfaces de datos. El SOM facilita la reutilización de la simulación y

proporciona un punto de partida para combinar la simulación con otras simulaciones participantes para un propósito común. El segundo es un modelo de objeto de federación (FOM). El FOM es una especificación que define la información intercambiada en tiempo de ejecución para lograr un conjunto dado de objetivos de federación. Esta información incluye clases de objeto, atributos de clase de objeto, clases de interacción, parámetros de interacción, contenido del Modelo de objetos de administración (MOM) y otra información relevante. El tercero es el MOM, que es un grupo de construcciones predefinidas que proporcionan soporte para monitorear y controlar la ejecución de una federación. Los FOM pueden desarrollarse desde cero utilizando elementos relevantes de los SOM de las aplicaciones federadas que conformarán la federación o utilizando los FOM existentes o los FOM de referencia y cualquier combinación de estos enfoques. Si se utilizan SOM o FOM existentes, es importante comprender que puede haber partes de estos SOM / FOM que pueden no ser relevantes para otras aplicaciones federadas en la federación y, por lo tanto, no se incorporarán en el FOM. Por ejemplo, una simulación de barco puede modelar la presión de la caldera, pero si una federación solo necesita conocer la ubicación del barco, la presión de la caldera puede caerse opcionalmente del FOM. Además, puede ser necesario conciliar diferentes unidades y diferentes representaciones entre simulaciones. El OMT proporciona una estructura estándar y un conjunto de convenciones de representación para especificar tanto SOM como FOM. (IEEE, 2010, pp: 3)

2.6.3.3. Grupos de servicio.

La especificación de interfaz federada define la interfaz funcional entre federados y el RTI. El RTI brinda servicios a federaciones unidas de manera similar a cómo un sistema operativo distribuido brinda servicios a las aplicaciones. Los servicios se organizan en siete grupos básicos (IEEE, 2010, Pág. 3):

a) **Gestión de la Federación.** Estos servicios permiten la coordinación de actividades de toda la federación a lo largo de la vida de la ejecución de una federación. Dichos servicios incluyen creación y destrucción de ejecución de

federación, unión y renuncia de aplicaciones federadas, puntos de sincronización de federación y operaciones de guardar y restaurar.

b) **Gestión de declaraciones.** Estos servicios permiten que los federados unidos especifiquen los tipos de datos que proporcionarán o recibirán de la ejecución de la federación. Este proceso se realiza a través de un conjunto de servicios de publicación y suscripción junto con algunos servicios relacionados.

c) **Gestión de objetos.** Estos servicios apoyan las actividades del ciclo de vida de los objetos e interacciones utilizados por los federados unidos de una ejecución de federación. Estos servicios permiten registrar y descubrir instancias de objetos, actualizar y reflejar los atributos de instancia asociados con estas instancias de objetos, eliminar o eliminar instancias de objetos, así como enviar y recibir interacciones y otros servicios relacionados. (Nota: las definiciones formales para cada uno de estos términos se pueden encontrar en la cláusula de definiciones de las tres especificaciones HLA).

d) **Gestión de la propiedad.** Estos servicios se utilizan para establecer el privilegio de una federación unida específica para proporcionar valores para un atributo de instancia de objeto, así como para facilitar la transferencia dinámica de este privilegio (propiedad) a otros federados unidos durante la ejecución de una federación.

e) **Gestión del tiempo.** Estos servicios permiten a las federaciones unidas operar con un concepto lógico de tiempo y mantener conjuntamente un reloj virtual distribuido. Estos servicios admiten simulaciones de eventos discretos y aseguran el orden causal entre los eventos.

f) **Gestión de distribución de datos.** Estos servicios permiten a los federados unidos especificar aún más las condiciones de distribución (más allá de las proporcionadas a través de los servicios de gestión de declaraciones) para los datos específicos que envían o solicitan recibir durante una ejecución de federación. El RTI utiliza esta información para enrutar los datos de los productores a los consumidores de una manera más personalizada.

g) **Servicios de apoyo.** Este grupo incluye servicios diversos utilizados por federaciones unidas para realizar acciones tales como transformaciones de nombre a nombre y de nombre a nombre, la configuración de interruptores de aviso, manipulaciones de región y el inicio y apagado de RTI.

2.6.3.4. Reglas.

Las reglas de HLA completan el marco. Requieren el uso de las especificaciones OMT y Federate Interface, así como también principios y prácticas de desarrollo sólidos para garantizar el funcionamiento adecuado de la ejecución de una federación. (IEEE, 2010, pp: 3)

2.6.3.5. Implementación independencia.

El HLA permite que los sistemas de simulación distribuida se desarrollen y ejecuten para una variedad de propósitos de manera estandarizada. El HLA no prescribe una implementación específica, ni exige el uso de ningún lenguaje o software de programación en particular. Con el tiempo, a medida que los avances tecnológicos estén disponibles, serán posibles implementaciones nuevas y diferentes en el marco del HLA. (IEEE, 2010, pp: 3)

2.6.3.6. Relación de HLA y conceptos orientados a objetos (OO).

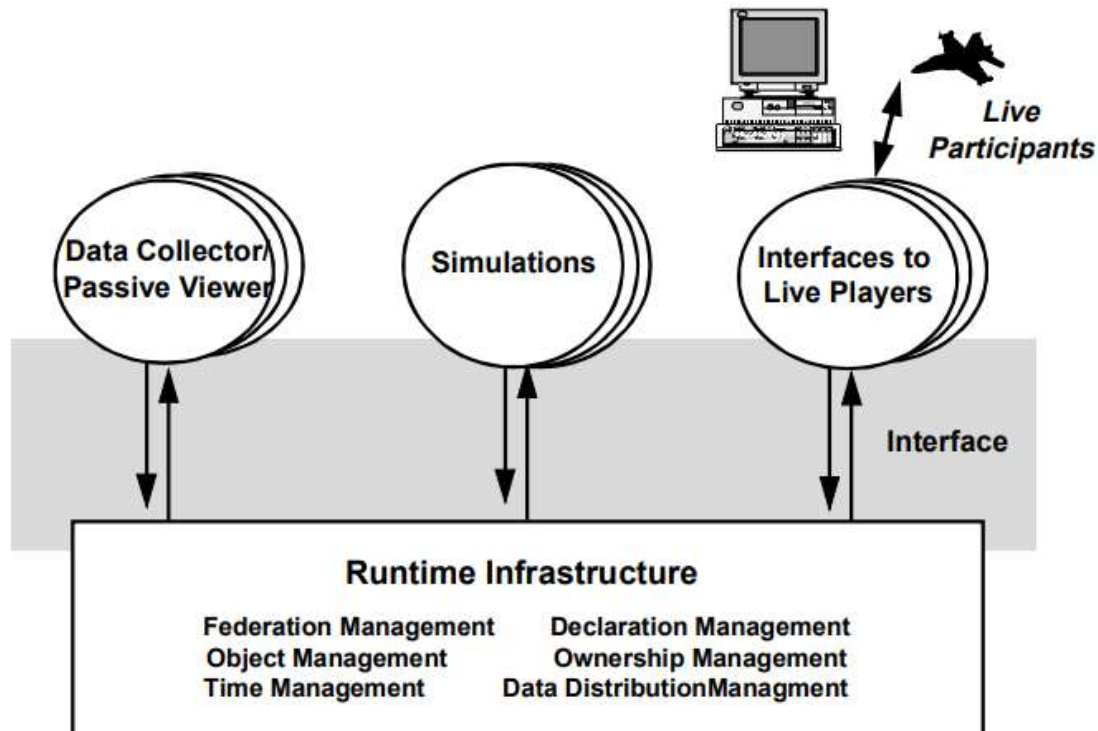
Aunque el HLA OMT es la estructura de documentación estandarizada para los modelos de objetos HLA, los FOM y los SOM no corresponden completamente a definiciones comunes de modelos de objetos en las técnicas de diseño y análisis de OO (OOAD). En la literatura de OOAD, un modelo de objeto se describe como una abstracción de un sistema desarrollado con el propósito de comprenderlo completamente. Para lograr esta comprensión, la mayoría de las técnicas de OO recomiendan definir varias vistas del sistema. Para los modelos de objetos HLA, el alcance previsto de la descripción del sistema es mucho más limitado y se centra específicamente en los requisitos y capacidades para el intercambio de información federada. Para los SOM, la intención es describir la interfaz pública de la federación en términos de un conjunto identificado de clases de objetos HLA compatibles y clases de interacción. Se debe proporcionar una descripción más completa de cómo se diseña y funciona internamente una federación (por ejemplo, un modelo de objeto

OO tradicional) a través de recursos de documentación distintos del SOM. Para los FOM, la intención es describir el intercambio de información que ocurre durante la ejecución de una federación. Las diferencias entre los principios y conceptos de HLA y OOAD también aparecen a nivel de objeto individual. En la literatura de OOAD, los objetos se definen como encapsulaciones de software de datos y operaciones (métodos). En el HLA, los objetos se definen completamente por las características de identificación (atributos), cuyos valores se intercambian entre federados durante la ejecución de una federación. Cualquier comportamiento y operaciones relacionados con OO que afecten los valores de los atributos del objeto HLA se mantienen residentes en las federaciones. Si bien las estructuras de clase de HLA están impulsadas por los requisitos de suscripción y las preocupaciones de crecimiento de FOM, las estructuras de clase en los sistemas OO generalmente están impulsadas por preocupaciones de eficiencia y mantenibilidad. Como se discutió anteriormente, las clases de objetos HLA se describen por los atributos que se definen para ellos. Estos atributos son, en términos generales, miembros de datos de la clase. Los atributos, que son propiedades abstractas de las clases de objetos HLA, se denominan atributos de clase. Las instancias de objeto HLA se generan a través de un servicio HLA utilizando una clase de objeto HLA como plantilla. Cada atributo contenido por una instancia de objeto HLA se denomina atributo de instancia. Los objetos OO interactúan mediante el paso de mensajes, en el que un objeto OO invoca una operación proporcionada por otro objeto OO. Los objetos HLA no interactúan directamente. Son los federados los que interactúan, a través de los servicios de HLA, actualizando los valores de los atributos de la instancia o enviando interacciones. Además, la responsabilidad de actualizar los atributos de instancia asociados con una instancia de objeto HLA puede distribuirse entre diferentes federados en una federación (distribuyendo efectivamente la responsabilidad de mantener el estado de la instancia de objeto HLA en toda la federación), mientras que los objetos OO encapsulan el estado localmente y asocian las responsabilidades de actualización con operaciones que están estrechamente vinculadas a la implementación del objeto en un lenguaje de programación OO.

2.6.3.7. HLA DEVELOPMENT PROCESS.

El HLA se desarrolló en base a un proceso que involucra al gobierno, la academia y la industria. En 1995, tres equipos de la industria desarrollaron conceptos para la definición de una arquitectura de alto nivel. Los resultados de estos esfuerzos se combinaron junto con información adicional de otros proyectos de modelado y simulación para llegar a la fase inicial. (Fujimoto, 1998, pp: 2)

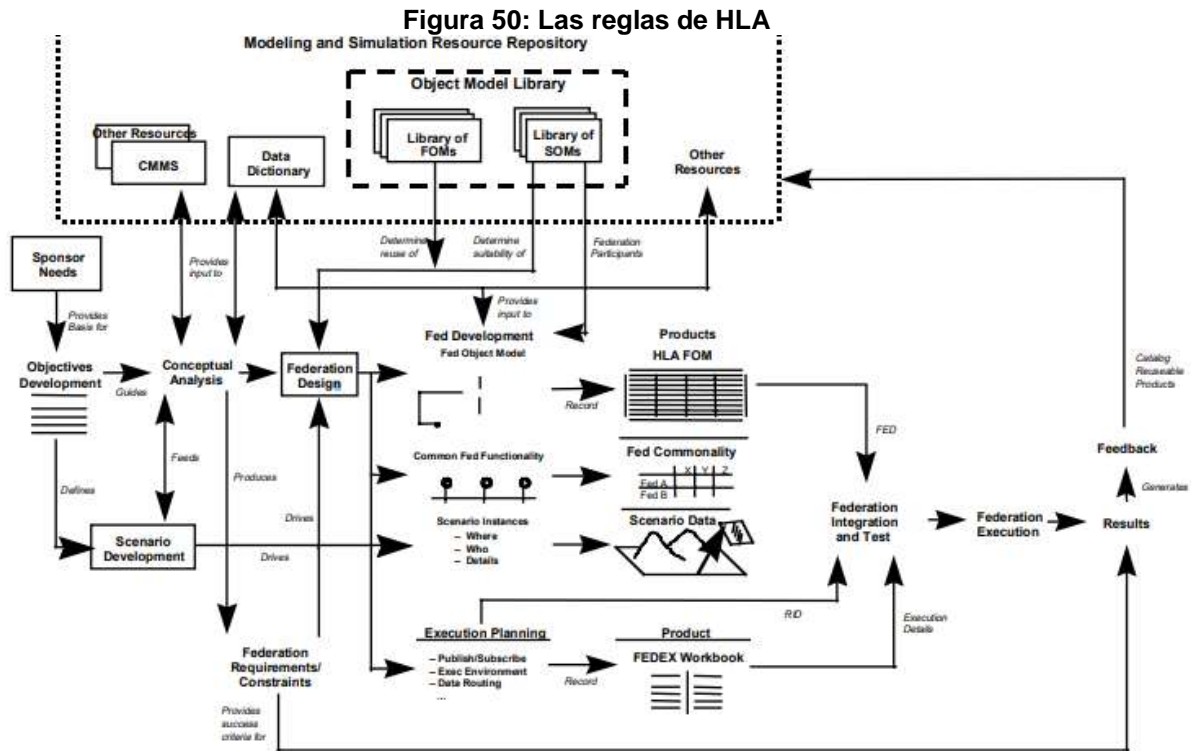
Figura 49: Proceso de desarrollo (HLA)



Fuente: Fujimoto, 1998

2.6.3.8. Reglas de HLA.

Finalmente, las reglas de HLA (Oficina de Modelado y Simulación de Defensa 1998) resumen los principios clave detrás de HLA. Las reglas se dividen en dos grupos: federación y reglas de federación. Las federaciones, o conjuntos de simulaciones o federaciones que interactúan, deben tener un FOM en el formato OMT. Durante el tiempo de ejecución, toda la representación de objetos tiene lugar en las federaciones (no en la RTI) con un solo federado que posee cualquier atributo dado de una instancia de un objeto en un momento dado. El intercambio de información entre los federados se realiza a través del RTI utilizando la especificación de interfaz HLA. (Fujimoto, 1998, pp: 7)



Fuente: Fujimoto, 1998

2.6.4. Quality assurance.

2.6.4.1. Fundamentos de calidad de software.

Llegar a un acuerdo sobre lo que constituye calidad para todos los interesados y comunicando claramente ese acuerdo con los ingenieros de software requiere que los muchos aspectos de la calidad se definan y discutan formalmente. Un ingeniero de software debe comprender conceptos de calidad, características, valores y su aplicación al software en desarrollo o mantenimiento. El concepto importante es que los requisitos del software definen los atributos de calidad requeridos del software. Los requisitos de software influyen en los métodos de medición y los criterios de aceptación para evaluar el grado en que el software y la documentación relacionada alcanzan los niveles de calidad deseados. (IEEE, 2014, pp. 175)

2.6.4.2. Calidad del proceso de software.

La gestión de la calidad del software y la calidad del proceso de ingeniería del software tienen una relación directa con la calidad del producto de software. Los modelos y criterios que evalúan las capacidades de las organizaciones de software

son principalmente consideraciones de organización y gestión de proyectos y, como tales, están cubiertos en la Gestión de Ingeniería de Software y los KA de Procesos de Ingeniería de Software. No es posible distinguir completamente la calidad del proceso de la calidad del producto porque los resultados del proceso incluyen productos. Determinar si un proceso tiene la capacidad de producir constantemente productos de la calidad deseada no es simple. El proceso de ingeniería de software, discutido en el Software Engineering Process KA, influye en las características de calidad de los productos de software, que a su vez afectan la calidad tal como la perciben las partes interesadas. (IEEE, 2014, pp: 177)

2.6.4.3. Verificación y validación.

El propósito de V&V es ayudar a la organización de desarrollo a incorporar calidad en el sistema durante el ciclo de vida. Los procesos V&V proporcionan una evaluación objetiva de los productos y procesos a lo largo del ciclo de vida. Esta evaluación demuestra si los requisitos son correctos, completos, precisos, consistentes y comprobables. Los procesos de V&V determinan si los productos de desarrollo de una actividad dada cumplen con los requisitos de esa actividad y si el producto satisface su uso previsto y las necesidades del usuario. La verificación es un intento de garantizar que el producto se construya correctamente, en el sentido de que los productos de salida de una actividad cumplen con las especificaciones impuestas en actividades anteriores. La validación es un intento de garantizar que se construya el producto correcto, es decir, que el producto cumpla su propósito específico. Tanto el proceso de verificación como el proceso de validación comienzan temprano en la fase de desarrollo o mantenimiento. Proporcionan un examen de las características clave del producto en relación tanto con el predecesor inmediato del producto como con las especificaciones que deben cumplirse. El propósito de planificar V&V es asegurar que cada recurso, rol y responsabilidad se asigne claramente. Los documentos del plan V&V resultantes describen los diversos recursos y sus roles y actividades, así como las técnicas y herramientas que se utilizarán. La comprensión de los diferentes propósitos de cada actividad de V&V ayuda en la planificación cuidadosa de las técnicas y recursos necesarios para cumplir con sus propósitos. El plan también aborda la gestión, la comunicación, las

políticas y los procedimientos de las actividades de V&V y su interacción, así como los informes de defectos y los requisitos de documentación. (IEEE, 2014, pp: 179)

2.6.4.4. Inspecciones.

Algunos diferenciadores importantes de las inspecciones como en comparación con otros tipos de revisiones técnicas son estas (IEEE, 2014, pp: 181):

1. Reglas. Las inspecciones se basan en el examen de un producto de trabajo con respecto a un conjunto definido de criterios especificados por la organización. Se pueden definir conjuntos de reglas para diferentes tipos de productos de trabajo.
2. Muestreo. Más bien ese intento de examinar cada palabra y figura en un documento, el proceso de inspección permite a los verificadores evaluar subconjuntos definidos (muestras) de los documentos bajo revisión.
3. Compañero. Las personas que ocupan puestos de dirección sobre los miembros del equipo de inspección no participan en la inspección. Esta es una distinción clave entre la revisión por pares y la revisión por la dirección.
4. Lead. Un moderador imparcial que está capacitado en técnicas de inspección dirige las reuniones de inspección.
5. Reunión. El proceso de inspección incluye reuniones (presenciales o electrónicas) realizadas por un moderador de acuerdo con un procedimiento formal en el que los miembros del equipo de inspección informan sobre las anomalías que han encontrado y otros problemas.

2.6.4.5. Técnicas estáticas.

Las técnicas estáticas examinan la documentación del software (incluidos los requisitos, las especificaciones de la interfaz, los diseños y los modelos) y el código fuente del software sin ejecutar el código. Existen muchas herramientas y técnicas para examinar estáticamente los productos de trabajo de software.

Las herramientas que analizan el flujo de control del código fuente y la búsqueda de código muerto se consideran herramientas de análisis estático porque no implican la ejecución del código del software. Otros tipos de técnicas analíticas más formales se

conocen como métodos formales. Se utilizan especialmente para verificar los requisitos y diseños de software. Se han utilizado principalmente en la verificación de partes cruciales de sistemas críticos, como requisitos específicos de seguridad y protección. (IEEE, 2014, pp: 182)

2.6.4.6. Técnicas Dinámicas.

Las técnicas dinámicas implican ejecutar el código del software. Se realizan diferentes tipos de técnicas dinámicas durante el desarrollo y mantenimiento del software. En general, se trata de técnicas de prueba, pero técnicas como la simulación y el análisis de modelos pueden considerarse dinámicas. La lectura de código se considera una técnica estática, pero los ingenieros de software experimentados pueden ejecutar el código a medida que lo leen. La lectura de códigos puede utilizar técnicas dinámicas. Esta discrepancia en la clasificación indica que las personas con diferentes roles y experiencia en la organización pueden considerar y aplicar estas técnicas de manera diferente. Diferentes grupos pueden realizar pruebas durante el desarrollo de software, incluidos grupos independientes del equipo de desarrollo. El Software Testing KA está dedicado por completo a este tema. (IEEE, 2014, pp: 183)

2.6.4.7. Testing.

Dos tipos de pruebas pueden incluirse en V&V debido a su responsabilidad por la calidad de los materiales utilizados en el proyecto:

- Evaluación y pruebas de herramientas para ser utilizadas en el proyecto.
- Pruebas de conformidad (o revisión de pruebas de conformidad) de componentes y productos COTS que se utilizarán en el producto.

A veces, una organización independiente (de terceros o IV&V) puede encargarse de realizar pruebas o monitorear el proceso de prueba. Se puede recurrir a V&V para evaluar las pruebas en sí: adecuación de planes, procesos y procedimientos, y adecuación y precisión de los resultados. El tercero no es el desarrollador, ni está asociado con el desarrollo del producto. En cambio, el tercero es una instalación independiente, generalmente acreditada por algún organismo de autoridad. Su

propósito es probar la conformidad de un producto con un conjunto específico de requisitos. (IEEE, 2014, pp: 184)

2.6.4.8. Pruebas de Estrés.

La prueba de estrés es el proceso de poner un sistema en condiciones extremas para verificar la robustez del sistema y / o para detectar varios problemas relacionados con la carga.

Ejemplos de tales condiciones pueden estar relacionados con la carga como sería el someter al sistema en condiciones normales o con una carga extremadamente pesada de recursos informáticos limitados (Ejemplo, Alto Uso del CPU) o fallas (por ejemplo, falla de la base de datos). En otros casos, las pruebas de estrés se utilizan para evaluar la eficiencia de los diseños de software. (Ahmed E. Hassan, 2015, pp: 1093)

2.6.4.9. Pruebas Unitarias.

Integre y pruebe componentes de software. La organización de implementación de software debe integrar y probar cada componente de software para asegurarse de que funcione como se espera. Las pruebas de integración de componentes no deben reconducir el software de pruebas unitarias, pero debe centrarse en garantizar que las interfaces de software funcionen correctamente y que no se introdujeron problemas al integrar elementos de software.

Pruebe las unidades de software. El equipo de implementación de software prueba cada unidad de software para asegurarse de que funcione como se espera y cumpla las especificaciones de la unidad de software. Los resultados deben documentarse en el informe de prueba de la unidad de software.

Las deficiencias identificadas durante las pruebas unitarias pueden resolverse, si es posible, mediante modificando el diseño programático y arreglando el código fuente apropiadamente. (Richard Schmidt, 2013, pp: 325)

Se deben generar informes de problemas de software para documentar las deficiencias identificadas que se derivan de las especificaciones de la unidad estructural.

Coloque las unidades de software bajo control de configuración. Una vez que una unidad de software satisfecho sus procedimientos de prueba unitaria, el código debe ser capturado en el software biblioteca y colocado bajo control de configuración.

2.6.4.10. Pruebas de integración.

Verificar que la implementación de software planifica, programa y paquetes de trabajo correctamente se ajustan a la configuración estructural. Esto implica el esfuerzo de diseñar, codificar y probar unidades estructurales y realizar pruebas e integración de software. Los paquetes de trabajo de montaje estructural deben revisarse para garantizar que se asignen recursos suficientes para la preparación y verificación de los talones de prueba del software. Diseño de unidades de software y las revisiones por pares deben proporcionar evidencia que verifique el cumplimiento de la unidad estructural.

Las revisiones por pares de integración de componentes deben proporcionar evidencia de que verifica el cumplimiento de las especificaciones de la unidad estructural. . (Richard Schmidt, 2013, pp: 317)

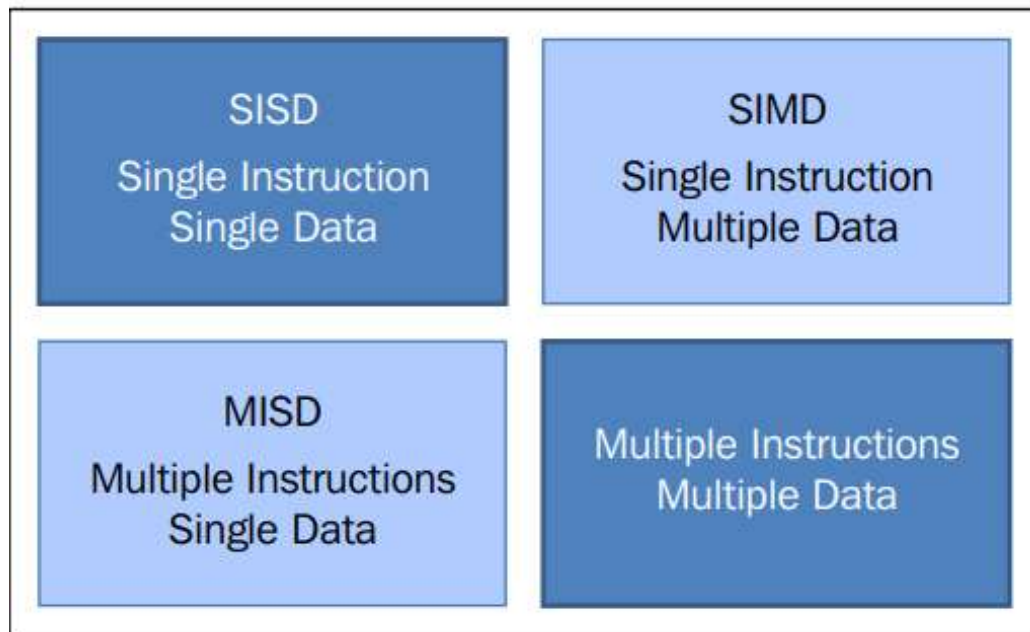
2.7. SISTEMAS DISTRIBUIDOS.

Según el número de instrucciones y datos que se pueden procesar simultáneamente, los sistemas informáticos se clasifican en cuatro categorías:

- Instrucción única, datos únicos (SISD)
- Instrucciones individuales, datos múltiples (SIMD)
- Instrucciones múltiples, datos únicos (MISD)
- Instrucciones múltiples, datos múltiples (MIMD)

Esta clasificación se conoce como taxonomía de Flynn. (Zaccone, 2015, Pág. 3)

Figura 51: Clasificación Flynn



Fuente: Zacccone, 2015

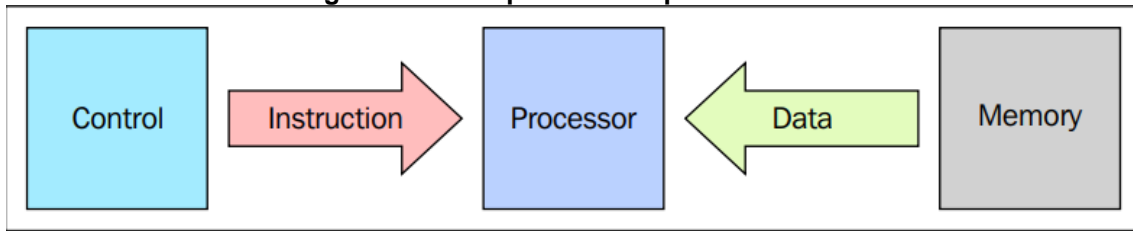
2.7.1. SISD

El sistema informático SISD es una máquina uniprocadora. Ejecuta una sola instrucción que opera en un solo flujo de datos. En SISD, las instrucciones de la máquina se procesan secuencialmente. (Zacccone, 2015, pp: 3)

En un ciclo de reloj, la CPU ejecuta las siguientes operaciones:

- Recuperar: la CPU obtiene los datos y las instrucciones de un área de memoria, que se denomina registro.
- Decodificar: la CPU decodifica las instrucciones.
- Ejecutar: la instrucción se lleva a cabo en los datos. El resultado de la operación se almacena en otro registro.
- Una vez que se completa la etapa de ejecución, la CPU se establece para comenzar otro ciclo de CPU.

Figura 52: El esquema de arquitectura SISD



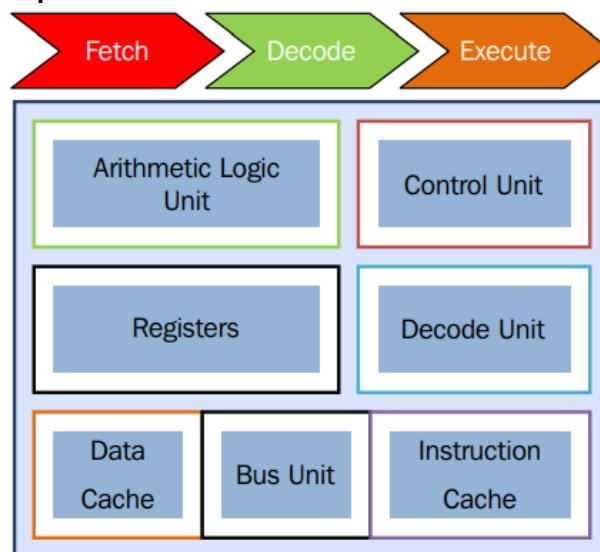
Fuente: Zaccone, 2015

Los algoritmos que se ejecutan en este tipo de computadoras son secuenciales (o seriales), ya que no contienen ningún paralelismo. Ejemplos de computadoras SISD son sistemas de hardware con una sola CPU. Los elementos principales de estas arquitecturas (arquitecturas de Von Neumann) son (Zaccone, 2015, pp: 4):

- Unidad de memoria central: se utiliza para almacenar tanto las instrucciones como los datos del programa.
- CPU: se utiliza para obtener las instrucciones y / o datos de la unidad de memoria, que decodifica las instrucciones y las implementa secuencialmente.
- El sistema de E / S: se refiere a los datos de entrada y salida del programa.

Las computadoras convencionales de un solo procesador se clasifican como sistemas SISD. La siguiente figura muestra específicamente qué áreas de una CPU se utilizan en las etapas de recuperación, decodificación y ejecución:

Figura 53: Componentes de la CPU en la fase FETCH-DECODE-EXECUTE

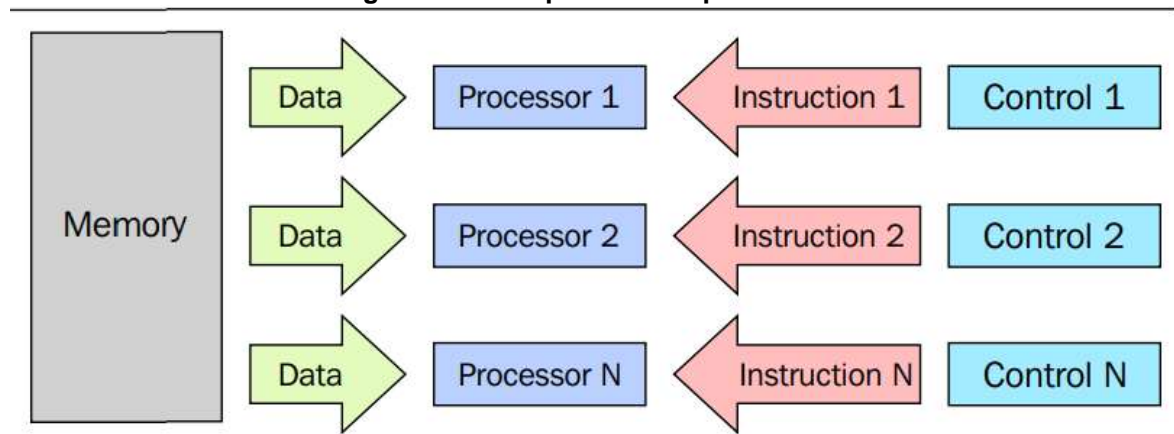


Fuente: Zaccone, 2015

2.7.2. MISD

En este modelo, n procesadores, cada uno con su propia unidad de control, comparten una sola unidad de memoria. En cada ciclo de reloj, todos los procesadores procesan los datos recibidos de la memoria simultáneamente, cada uno de acuerdo con las instrucciones recibidas de su unidad de control. En este caso, el paralelismo (paralelismo a nivel de instrucción) se obtiene al realizar varias operaciones en el mismo dato. Los tipos de problemas que se pueden resolver de manera eficiente en estas arquitecturas son bastante especiales, como los relacionados con el cifrado de datos; Por esta razón, la computadora MISD no encontró espacio en el sector comercial. Las computadoras MISD son más un ejercicio intelectual que una configuración práctica. (Zaccone, 2015, pp: 5)

Figura 54: El esquema de arquitectura MISD



Fuente: Zaccone, 2015

2.7.3. SIMD

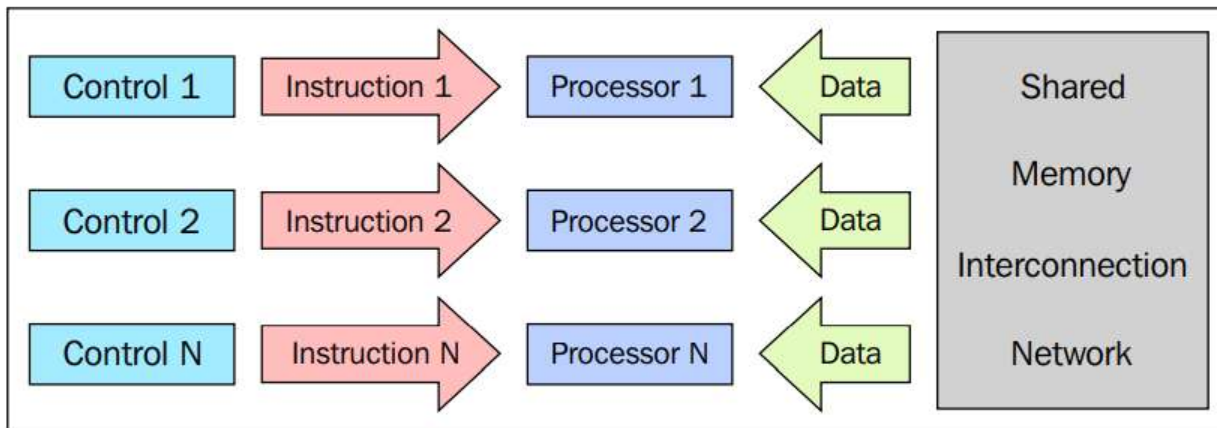
Una computadora SIMD consta de n procesadores idénticos, cada uno con su propia memoria local, donde es posible almacenar datos. Todos los procesadores funcionan bajo el control de una sola secuencia de instrucciones; Además de esto, hay n flujos de datos, uno para cada procesador. Los procesadores trabajan simultáneamente en cada paso y ejecutan la misma instrucción, pero en diferentes elementos de datos. Este es un ejemplo de paralelismo a nivel de datos. Las arquitecturas SIMD son mucho más versátiles que las arquitecturas MISD. Numerosos problemas que cubren una amplia gama de aplicaciones pueden resolverse mediante algoritmos paralelos en computadoras SIMD. Otra característica interesante es que los algoritmos para

estas computadoras son relativamente fáciles de diseñar, analizar e implementar. El límite es que solo los problemas que se pueden dividir en varios subproblemas (que son todos idénticos, cada uno de los cuales se resolverá simultáneamente, a través del mismo conjunto de instrucciones) se pueden abordar con la computadora SIMD. Con la supercomputadora desarrollada de acuerdo con este paradigma, debemos mencionar Connection Machine (1985 Thinking Machine) y MPP (NASA - 1983). Programación de GPU con Python, el advenimiento de la moderna unidad de procesador de gráficos (GPU), construida con muchas unidades integradas SIMD, ha llevado a un uso más extendido de este paradigma computacional. (Zacccone, 2015, pp: 6)

2.7.4. MIMD

Esta clase de computadoras paralelas es la clase más general y más poderosa según la clasificación de Flynn. Hay n procesadores, n secuencias de instrucciones y n secuencias de datos en esto. Cada procesador tiene su propia unidad de control y memoria local, lo que hace que las arquitecturas MIMD sean computacionalmente más potentes que las utilizadas en SIMD. Cada procesador opera bajo el control de un flujo de instrucciones emitidas por su propia unidad de control; por lo tanto, los procesadores pueden potencialmente ejecutar diferentes programas en diferentes datos, resolviendo subproblemas que son diferentes y pueden ser parte de un solo problema mayor. En MIMD, la arquitectura se logra con la ayuda del nivel de paralelismo con hilos y / o procesos. Esto también significa que los procesadores suelen funcionar de forma asíncrona. Las computadoras de esta clase se utilizan para resolver aquellos problemas que no tienen una estructura regular que requiere el modelo SIMD. Hoy en día, esta arquitectura se aplica a muchas PC, supercomputadoras y redes de computadoras. Sin embargo, hay un contador que debe tener en cuenta: los algoritmos asíncronos son difíciles de diseñar, analizar e implementar. (Zacccone, 2015, pp: 6)

Figura 55: El esquema de arquitectura MIMD

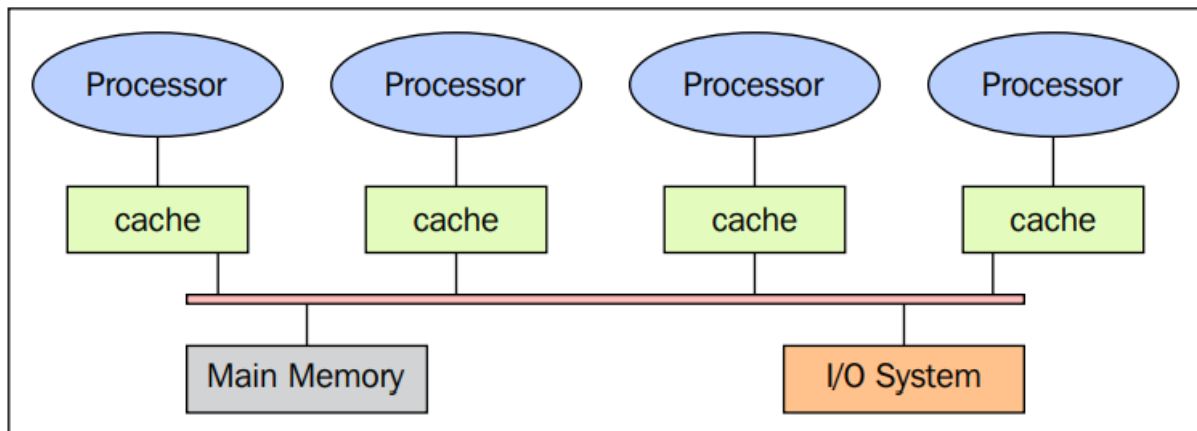


Fuente: Zacccone, 2015

2.7.5. Memoria compartida

El esquema de un sistema multiprocesador de memoria compartida se muestra en la siguiente figura. Las conexiones físicas aquí son bastante simples. La estructura del bus permite un número arbitrario de dispositivos que comparten el mismo canal. Los protocolos de bus se diseñaron originalmente para permitir que un único procesador y uno o más discos o controladores de cinta se comuniquen a través de la memoria compartida aquí. Tenga en cuenta que cada procesador se ha asociado con una memoria caché, ya que se supone que la probabilidad de que un procesador necesite datos o instrucciones presentes en la memoria local es muy alta. El problema ocurre cuando un procesador modifica los datos almacenados en el sistema de memoria que otros procesadores usan simultáneamente. El nuevo valor pasará del caché del procesador que se ha cambiado a la memoria compartida; más tarde, sin embargo, también debe pasarse a todos los demás procesadores, para que no funcionen con el valor obsoleto. Este problema se conoce como el problema de la coherencia de la memoria caché, un caso especial del problema de la consistencia de la memoria, que requiere implementaciones de hardware que pueden manejar problemas de concurrencia y sincronización similares a los que tienen programación de subprocesos. (Zacccone, 2015, pp: 7)

Figura 56: El esquema de arquitectura de memoria compartida



Fuente: Zaccane, 2015

2.8. BASES DE DATOS

2.8.1. Creación de tablas en SQL

SQL respalda tres tipos de tablas: tablas base, tablas derivadas y tablas vistas. La mayoría de las tablas base son objetos de esquema que tienen los datos de SQL. Las tablas derivadas son los resultados que se observan cuando se solicitan (consultan) datos de una base de datos. Se puede pensar en las tablas vistas como un tipo de nombre derivado de una tabla, con una definición de la vista almacenado en el esquema:

- CREATE TABLE
- CREATE GLOBAL TEMPORARY TABLE
- CREATE LOCAL TEMPORARY TABLE

Dependiendo del RDBMS en que trabaje, puede que tenga que calificar el nombre de la tabla incluyendo un nombre de esquema, identificador autorizado, o el nombre de la base de datos (por ejemplo, INVENTARIO.ARTISTAS). (Opel, 2009, pp: 52)

2.8.2. Especificación de los tipos de datos en una columna

Cada vez que se define una columna en la instrucción CREATE TABLE, al menos debe proporcionar un nombre para la columna y un tipo de dato asociado o un dominio. El tipo de datos o dominio limita los valores que pueden introducirse en esa columna. Por ejemplo, algunos tipos de datos limitan los valores de una columna a

números, mientras otros tipos de datos permiten que se introduzca cualquier carácter. SQL respalda tres formas de tipos de datos (Opel, 2009, pp: 52-54):

- **Predefinido** Los tipos de datos predefinidos son los más comunes. Cada tipo de datos predefinido es un elemento nombrado (utilizando una palabra clave en SQL) que limita valores a las restricciones definidas por esa base de datos. SQL incluye cinco formas de tipos de datos predefinidos: cadena, numérico, fecha y hora, intervalo y booleano.
- **Construido** Los tipos de datos contruidos también se denominan elementos, pero tienden a ser más complejos que los tipos de datos predefinidos ya que pueden contener múltiples valores. Los tipos contruidos permiten construir estructuras más complicadas que la mayoría de los tipos de datos tradicionales.
- **Definido por el usuario** Los tipos de datos definidos por el usuario se basan en los tipos predefinidos o definiciones de atributos, y se agregan como objetos de esquema al entorno SQL. SQL respalda dos formas de tipos de datos definidos por el usuario: distinto y estructurado. El tipo distinto se basa en el tipo de dato predefinido, y el tipo estructurado se basa en las definiciones de atributo. Se analizarán los tipos definidos por el usuario en la sección “Creación de tipos definidos por el usuario”.

2.8.2.1. Tipos de datos de cadena

Los tipos de datos de cadena se componen por tipos que permiten valores basados en los conjuntos de caracteres o en bits de datos. Los valores permitidos por los tipos en cadena pueden ser de longitud fija o variable, dependiendo del tipo específico. SQL define cuatro formas de tipos de datos en cadena (Opel, 2009, pp: 57):

- **Cadenas de caracteres** Los valores permitidos se deben extraer de un conjunto de caracteres, ya sea de un conjunto predeterminado o de un conjunto definido en el momento que la columna se define. Los tipos de datos

en la cadena de caracteres incluyen CHARACTER, CHARACTER VARYING y CHARACTER LARGE OBJECT.

- Cadenas de carácter nacional Los valores permitidos son similares a los de las cadenas de caracteres, salvo que el conjunto de caracteres asociados con estos tipos de datos se define por la aplicación. Como resultado, cuando una cadena de carácter nacional se especifica, los valores asociados con ese tipo de datos deben basarse en el conjunto de caracteres especificado por el sistema de gestión de base de datos relacional (RDBMS) para las cadenas de carácter nacional. Éstas son útiles para almacenar cadenas de caracteres en varios lenguajes en la misma base de datos. Los tipos de datos en la cadena de carácter nacional incluyen NATIONAL CHARACTER, NATIONAL CHARACTER VARYING y NATIONAL CHARACTER LARGE OBJECT.
- Cadenas de bits Los valores permitidos se basan en bits de datos (dígitos binarios), en lugar de conjuntos de caracteres o cotejos, lo que significa que estos tipos de datos permiten sólo valores de 0 o 1. SQL respalda dos formas de tipos de datos de cadena de bits: BIT y BIT VARYING.
- Cadenas binarias Los valores permitidos son similares a las cadenas de bits, excepto que se basan en bytes (denominado como octetos en SQL:2006), en lugar de bits. Como resultado, ningún conjunto de caracteres o cotejos se relacionan con ellas. (Un byte es igual a 8 bis, la razón por la cual el estándar SQL utiliza el término octeto.) SQL sólo respalda un tipo de datos en una cadena binaria: BINARY LARGE OBJECT. Este tipo es útil para almacenar datos binarios puros tales como clips de sonido o imágenes en la base de datos. Ahora que tiene una descripción general de las formas de tipos de datos en cadena, echemos un vistazo más de cerca a cada uno.

a. Tipos de datos numéricos

Como probablemente podrá suponer por el nombre, los valores especificados por los tipos de datos numéricos son números. Todos los tipos de datos numéricos tienen una precisión, y algunos tienen una escala. La precisión se refiere al número de dígitos (dentro de un valor numérico específico) que se pueden almacenar. La escala se refiere al número de dígitos en la parte fraccional de ese valor (los dígitos a la

derecha del punto decimal). Por ejemplo, el número 435.27 tiene una precisión de 5 y una escala de 2. La escala no puede ser un número negativo o ser más larga que la precisión. Una escala de 0 indica que el número es un número entero y no contiene ningún componente fraccional. SQL define dos formas de tipos de datos numéricos:

- **Numéricos exactos** Los valores permitidos tienen precisión y escala, que, para algunos tipos de datos numéricos, se definen por la implementación. Los tipos de datos numéricos exactos incluyen NUMERIC, DECIMAL, INTEGER y SMALLINT.
- **Numéricos aproximados** Los valores permitidos tienen precisión, pero no escala. Como resultado, el punto decimal puede flotar. Un número de punto flotante es aquel que contiene un punto decimal, pero el punto decimal se puede localizar en cualquier lugar dentro del número, por lo que se dice que un numérico aproximado no tiene escala. Los tipos de datos numéricos aproximados incluyen REAL, DOUBLE PRECISION y FLOAT.

2.8.2.2. Tipos de datos de fecha y hora

Como su nombre lo indica, los tipos de datos de fecha y hora se refieren a las formas de seguimiento de fechas y horas. SQL define tres formas de tipos de datos de fecha y hora (DATE, TIME y TIMESTAMP) y las variaciones en estos tipos. Estas variaciones están relacionadas con el Tiempo Universal Coordinado (UTC), que solía llamarse el Tiempo Medio de Greenwich (GMT), y las distintas zonas horarias. (Opel, 2009, pp: 58)

2.8.2.3. Tipos de datos de intervalo

El tipo de datos de intervalo está estrechamente relacionado con los tipos de datos de fecha y hora. El valor de un tipo de datos de intervalo representa la diferencia entre dos valores de fecha y hora. SQL respalda dos tipos de intervalos:

- **Intervalos de año-mes** El tipo de datos de intervalo especifica intervalos entre años, meses, o ambos. Se pueden utilizar sólo los campos de AÑO y MES en un intervalo de año-mes.

- Intervalos de día-hora El tipo de datos de intervalo especifica intervalos entre cualquiera de los siguientes valores: días, horas, minutos o segundos. Se pueden utilizar sólo los campos de DAY, HOUR, MINUTE y SECOND en un intervalo de día-tiempo.

No se puede mezclar un tipo de intervalo con los demás. Por ejemplo, no se puede definir un tipo de datos de intervalo que utilice el campo YEAR y el campo HOUR. El tipo de datos de intervalo utiliza la palabra clave INTERVAL seguido por una cláusula de. La cláusula es una serie de reglas complejas que describen cómo se puede definir el tipo de datos de INTERVAL para expresar la participación de los intervalos de años, meses, días, horas, minutos o segundos. Además, el campo principal (la primera palabra) en la cláusula se puede definir con una precisión (p). La precisión es el número de dígitos que se utilizan en el campo principal. Si la precisión no se especifica, la precisión predeterminada es 2. Para los intervalos de año-mes, se puede especificar uno de los siguientes tipos de datos de intervalo (Opel, 2009, Pág. 60):

- INTERVAL YEAR
- INTERVAL YEAR(p)
- INTERVAL MONTH
- INTERVAL MONTH(p)
- INTERVAL YEAR TO MONTH
- INTERVAL YEAR(p) TO MONTH

Hay muchas más opciones para los intervalos día-hora, ya que hay más campos de dónde escoger. Por ejemplo, se pueden especificar cualesquiera de los siguientes tipos de intervalos utilizando el campo DAY como campo principal o independientemente del campo:

- INTERVAL DAY
- INTERVAL DAY(p)
- INTERVAL DAY TO HOUR
- INTERVAL DAY(p) TO HOUR

- INTERVAL DAY TO MINUTE
- INTERVAL DAY(p) TO MINUTE
- INTERVAL DAY TO SECOND
- INTERVAL DAY(p) TO SECOND
- INTERVAL DAY TO SECOND(x)
- INTERVAL DAY(p) TO SECOND(x)

Cuando el campo de rastreo (la última palabra) es SECOND, se puede especificar una precisión adicional (x), que define el número de dígitos después del punto decimal. Como se puede observar de estos ejemplos, hay muchos más tipos de datos de intervalos de día-hora que se pueden definir. Sin embargo, tenga en cuenta que el campo principal debe tener siempre una unidad de tiempo superior que el campo de rastreo. Por ejemplo, el campo YEAR es mayor que MONTH, y HOUR es mayor que MINUTE. Si fuera a utilizarse un tipo de datos de intervalo en una definición de columna, se vería algo como lo siguiente:

- RANGO_FECHA INTERVAL YEAR(4) TO MONTH

2.8.2.4. Tipos de datos booleanos

El tipo de datos booleano (a diferencia de los tipos de datos de intervalo) es muy sencillo y fácil de aplicar. El tipo de dato respalda una construcción de verdadero/falso que permite sólo tres valores: verdadero, falso o desconocido. Un valor nulo se evalúa como desconocido. Los valores en el tipo de datos booleano se pueden utilizar en consultas de SQL y expresiones con fines de comparación. Las comparaciones en el tipo de datos booleano siguen una lógica específica:

- Verdadero es mayor que falso.
- Una comparación con un valor desconocido (nulo) devolverá un resultado desconocido.
- Un valor desconocido se puede asignar a una columna sólo si admite valores nulos.

Para utilizar el tipo de datos booleano, debe utilizar la palabra clave BOOLEAN sin parámetros, como se muestra en el siguiente ejemplo: ARTISTAS_CON_AGENTE BOOLEAN La columna ARTISTAS_CON_AGENTE sólo aceptará valores de verdadero, falso y desconocido. (Opel, 2009, pp: 62).

CAPÍTULO III: MARCO PRÁCTICO



CAPITULO 3: MARCO PRÁCTICO

3.1. ANÁLISIS DEL PROCESO ACTUAL DE ENTRENAMIENTO CON FUSIL Y SU ENTORNO DE DESEMPEÑO.

El siguiente análisis permitirá el entendimiento de los procesos de entrenamiento que se realizan en los diferentes cuarteles militares de Bolivia y las características climatológicas donde se realizan las prácticas de tiro con fusil.

3.1.1. Recopilación de información del proceso entrenamiento de tiro con fusil en el ejército boliviano.

Para la recolección de información sobre el proceso de entrenamiento de tiro con fusil se utilizaron las técnicas de recolección de información.

Aplicando una entrevista semiestructurada al Cnl. DAEN. Tomas Crisner Prado. En la cual detalló las fases que conlleva este entrenamiento. La entrevista puede ser encontrada en el Anexo C.

3.1.2. Recopilación de información sobre los fusiles utilizados en los entrenamientos de tiro con fusil.

Para obtener información relevante se realizó una entrevista al My. DIM. Camilo Rios Aguilar, donde se pudo determinar modelos de fusiles de asalto que forma parte de los requerimientos funcionales.

Así también se hizo un análisis de la documentación obtenida de la institución especialista en el tema, donde se pudo recabar información técnica respecto a estos fusiles. Ver Cuadro 1, Cuadro 2 y Cuadro 3.

En base a la información proporcionada por el My. DIM. Camilo Rios Aguilar se pudo extraer lo siguiente.

En toda la región altiplánica y dentro del margen de la segunda división y décima división emplean como estándar el fusil de asalto GALIL AR y FN FAL. En cuanto a la séptima división y octava división emplean el fusil de asalto AK-47 y FN FAL.

Sus especificaciones técnicas de los fusiles mencionados estarán detalladas en los siguientes cuadros:

Cuadro 1: Características técnicas del fusil de asalto AK-47

Origen	Rusia.
Calibre	7.62 x 51mm.
Dimensiones	868,00 mm.
Peso (sin munición)	4.3 Kg.
Alcance efectivo	400 m.
Sistema	Recuperación por toma de gases en un punto del cañón.
Velocidad de salida	700 m/seg.
Sistema de puntería	Alza regulable en distancia y guión.
Capacidad del cargador	30,40 y hasta 75 cartuchos, dependiendo del cargador.
Cadencia	600 dpm.
Variantes	AKS, AKM, RPK, AK12, etc.

Fuente: Elaboración propia, 2020.

En el cuadro 1 se puede observar las características más relevantes a tomar en cuenta del fusil de asalto AK-47, detalles como el alcance, velocidad de salida y cadencia.

Cuadro 2: Características técnicas del fusil de asalto FN FAL

Origen	Bélgica.
Calibre	7.62 x 51mm.

Dimensiones	1016mm.
Peso (sin munición)	4.2 Kg.
Alcance efectivo	600 m.
Sistema	Acción directa de los gases por toma de los mismos en la punta del cañón.
Velocidad de salida	835 m/seg.
Régimen de fuego	Semiautomático y automático.
Capacidad del cargador	20 cartuchos.
Cadencia	650 dpm.
Variantes	FAL PARA, FAL FAP.

Fuente: Elaboración propia, 2020.

En el cuadro 2 y 3 se puede observar la información más relevante de los fusiles FN FAL y GALIL AR, resaltando el alcance efectivo, velocidad de salida y cadencia.

Cuadro 3: Características técnicas del fusil de asalto GALIL AR

Origen	Israel.
Calibre	5.56 x 45 mm.
Dimensiones	1050 mm.
Peso (sin munición)	3.95 Kg.
Alcance efectivo	500 m.
Sistema de enfriamiento	Por aire.
Velocidad de salida	915 m/s.
Régimen de fuego	Automático, con retroceso por acción de los gases.
Capacidad del cargador	35 cartuchos.

Cadencia	600 dpm.
Variantes	Galil ARM, Galil SAR, Galil MAR, etc.

Fuente: Elaboración propia, 2020.

En conclusión, se pudo obtener todas las variables características de los fusiles utilizados para los entrenamientos de tiro con fusil los cuales nos permitirán más adelante facilitar la implementación de estos en los entornos virtuales.

3.1.3. Recolección de información sobre los diferentes entornos geográficos donde se realizan los entrenamientos de tiro con fusil.

Actualmente en Cochabamba, Cercado, los entrenamientos de tiro con fusil se realizan en el polígono de tiro ubicado en Cotapachi dentro del Regimiento de Artillería 7 Tumusla, el cual cuenta con dos polígonos de tiro, uno con una inclinación leve que es utilizada por la Fábrica Boliviana de Municiones (FBM), y el segundo con una inclinación mayor a la del primer polígono utilizado por los distintos cuarteles para los entrenamientos de tiro con fusil.

El polígono de tiro mencionado es en un campo abierto, con una edificación simple en los puntos de disparo. La zona donde se realizan los entrenamientos de tiro con fusil tiene una vegetación leve y está alejada de población civil, a continuación, se mostrará una imagen del polígono de tiro en Cotapachi.

Figura 57: Polígono de tiro utilizado por la fábrica boliviana de municiones FBM.



Fuente: Soto Diana. 2017.

Siguiente a esto por medio Google maps podemos tener una vista satelital de los dos polígonos de tiro, con las coordenadas (-17.418555, -66.260514) se puede apreciar lo siguiente:

Figura 58: Polígonos de tiro en Cotapachi



Fuente: Elaboración propia, 2020.

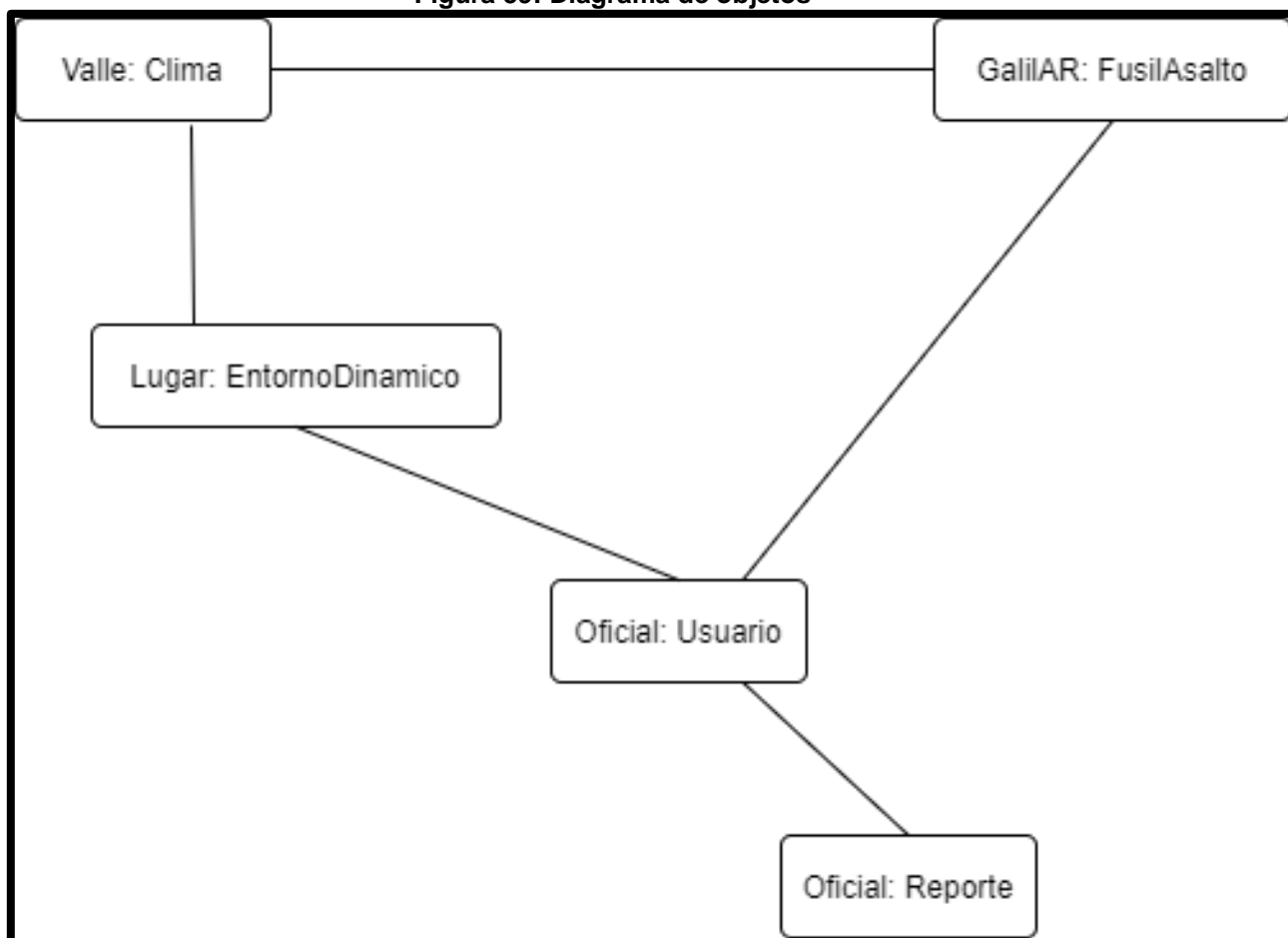
3.2. APLICACIÓN DEL MÉTODO FDD (FEATURE-DRIVEN-DEVELOPMENT) PARA DESARROLLAR EL SIMULADOR SEGÚN LA ARQUITECTURA HLA (HIGH LEVEL ARCHITECTURE).

Se llevará a cabo la aplicación del método ágil FDD, Que nos permitirá estructurar de una manera general, lo que se realizará para poder iniciar con el desarrollo del simulador tomando en cuenta los rasgos generales del simulador. El desglose del método, se lo realizará de manera detallada en los siguientes puntos.

3.2.1. Realización de un diagrama de objetos sobre el funcionamiento del simulador para el entrenamiento de tiro con fusiles.

Tomando en cuenta los aspectos a grandes rasgos del simulador se realizó el siguiente diagrama de objetos:

Figura 59: Diagrama de objetos



Fuente: Elaboración propia, 2020.

Siendo que este diagrama de objetos es una representación general de todo lo que implica el simulador, el cual sirvió para dar idea de lo que se tendría que realizar.

En base a este diagrama se comenzó con el desarrollo del simulador de tiro con fusiles de asalto y el diagrama general fue siendo complementado al punto de llegar a un diagrama de objetos mucho más completo.

3.2.2. Construcción de una lista de características del simulador para el entrenamiento de tiro con fusiles.

Se procederá a construir la lista de características del simulador para el entrenamiento de tiro con fusiles de manera general, durante la implementación puede que esta lista pueda ir variando de contenido.

Cuadro 4: Lista de características del simulador para el entrenamiento de tiro con fusiles

1	Obtención de imagen satelital en escala de grises.
2	Aplicación del método procedural para la generación del terreno.
3	Agregar variables climatológicas al entorno virtual generado.
4	Agregar variables y fórmulas físicas para la trayectoria de los proyectiles.
5	Generar una base de datos para almacenar reportes.
6	Programar las vistas de la BD.
7	Programar la funcionalidad para la generación de reportes por usuario.
8	Modelado 3d del fusil de asalto AK-47.
9	Modelado 3d del fusil de asalto Galil AR.
10	Modelado 3d del fusil de asalto FN FAL.
11	Modelado 3d de la munición 5.56x45mm.
12	Modelado 3d de la munición 7.62x51mm.
13	Aplicado de texturas a los modelos 3D generados.
14	Diseño y aplicación de animaciones para los fusiles de asalto.

Fuente: Elaboración propia, 2020.

3.2.3. Agrupación de las características del simulador para el entrenamiento de tiro con fusiles en áreas subjetivas.

Para realizar la agrupación de las características, se lo diferenciarán por colores, y relacionando en temas subjetivos.

Cuadro 5: Lista de características del simulador para el entrenamiento de tiro con fusiles

1	Obtención de imagen satelital en escala de grises.
2	Aplicación del método procedural para la generación del terreno.
3	Agregar variables climatológicas al entorno virtual generado.
4	Agregar variables y fórmulas físicas para la trayectoria de los proyectiles.
5	Generar una base de datos para almacenar reportes.
6	Programar las vistas de la BD.
7	Programar la funcionalidad para la generación de reportes por usuario.
8	Modelado 3d del fusil de asalto AK-47.
9	Modelado 3d del fusil de asalto Galil AR.
10	Modelado 3d del fusil de asalto FN FAL.
11	Modelado 3d de la munición 5.56x45mm.
12	Modelado 3d de la munición 7.62x51mm.
13	Aplicado de texturas a los modelos 3D generados.
14	Diseño y aplicación de animaciones para los fusiles de asalto.

Fuente: Elaboración propia, 2020.

3.2.4. Planificación en base de cada característica del simulador para el entrenamiento de tiro con fusiles.

Para la parte de planificación se tomará como base las características que generamos en el anterior punto, la cual será un cuadro que contendrá el número de característica, el nombre de la característica y la dificultad evaluada de 0 a 10, la dificultad estará dada en base al criterio propio del Chief Programmer o jefe de programación.

Cuadro 6: Características del simulador

Lista de características del simulador para el entrenamiento de tiro con fusiles		Dificultad (0 a 10)
1	Obtención de imagen satelital en escala de grises.	5
2	Aplicación del método procedural para la generación del terreno.	9
3	Agregar variables climatológicas al entorno virtual generado.	7
4	Agregar variables y fórmulas físicas para la trayectoria de los proyectiles.	6
5	Generar una base de datos para almacenar reportes.	4
6	Programar las vistas de la BD.	5
7	Programar la funcionalidad para la generación de reportes por usuario.	5
8	Modelado 3d del fusil de asalto AK-47.	6
9	Modelado 3d del fusil de asalto Galil AR.	6
10	Modelado 3d del fusil de asalto FN FAL.	6

11	Modelado 3d de la munición 5.56x45mm.	4
12	Modelado 3d de la munición 7.62x51mm.	4
13	Aplicado de texturas a los modelos 3D generados.	5
14	Diseño y aplicación de animaciones para los fusiles de asalto.	8

Fuente: Elaboración propia, 2020.

Para tomar en cuenta el esfuerzo determinado por la cantidad dada en la columna de dificultad, utilizamos una burn-up chart para el seguimiento de las iteraciones, para poder concluir el simulador a su totalidad sin ningún contratiempo. Cada sprint consta de un tiempo de dos semanas.

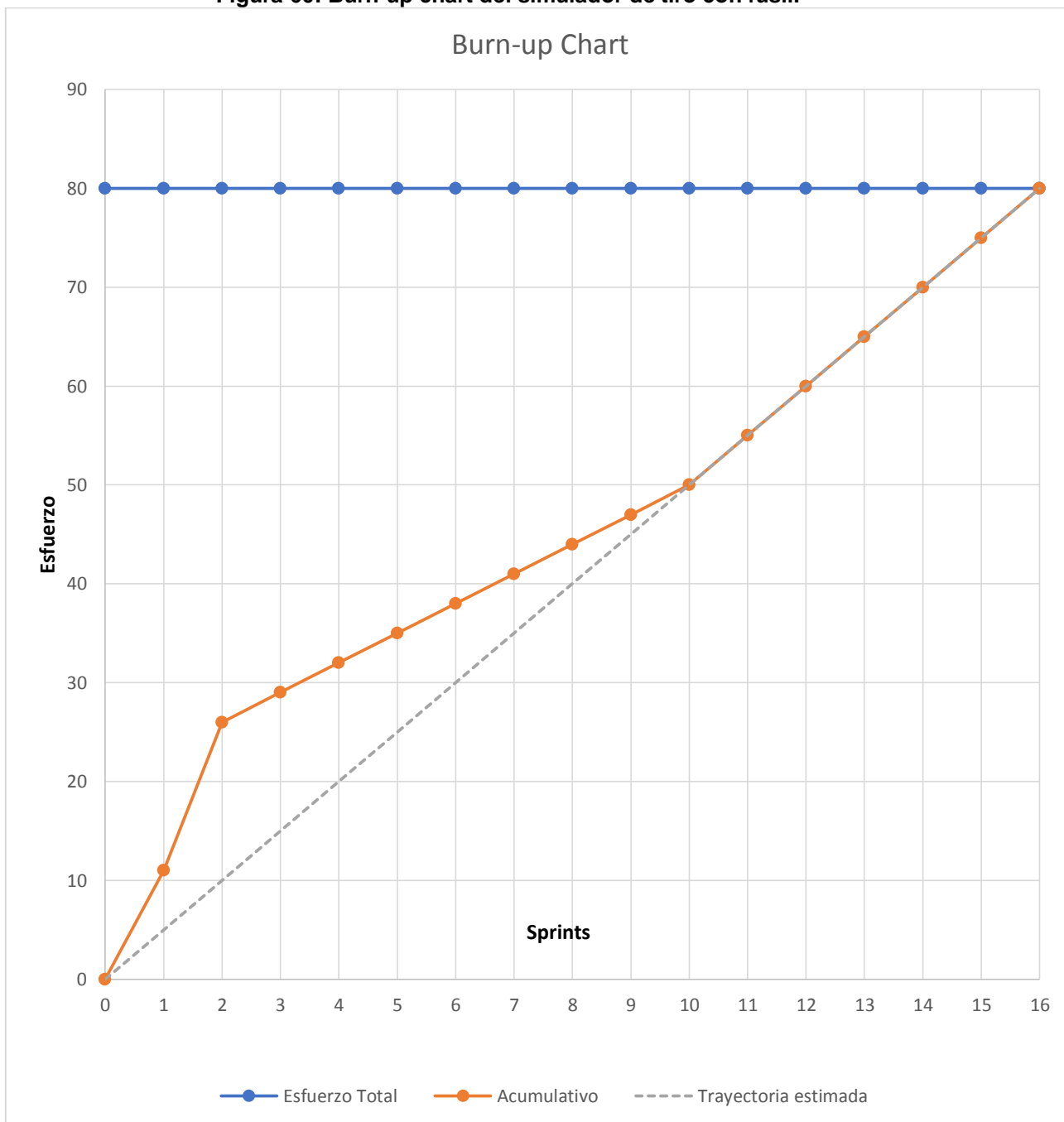
Cuadro 7: Esfuerzo del simulador de tiro.

IMPLEMENTACIÓN DEL TERRENO Y CARACTERÍSTICAS DE DISPARO.	Esfuerzo Estimado
Obtención de imagen satelital en escala de grises.	5
Aplicación del método procedural para la generación del terreno.	9
Agregar variables climatológicas al entorno virtual generado.	7
Agregar variables y fórmulas físicas para la trayectoria de los proyectiles.	6
IMPLANTACIÓN DE UNA TABLA ESTADÍSTICA DE TIRO.	
Generar una base de datos para almacenar reportes.	4
Programar las vistas de la BD.	5
Programar la funcionalidad para la generación de reportes por usuario.	5
MODELADO EN 3D DE LOS FUSILES PROPUESTOS.	
Modelado 3d del fusil de asalto AK-47.	6
Modelado 3d del fusil de asalto Galil AR.	6
Modelado 3d del fusil de asalto FN FAL.	6
Modelado 3d de la munición 5.56x45mm.	4
Modelado 3d de la munición 7.62x51mm.	4
Aplicado de texturas a los modelos 3D generados.	5
Diseño y aplicación de animaciones para los fusiles de asalto.	8
Total esfuerzo	80

Fuente: Elaboración propia, 2020.

Utilizando los valores del Cuadro 7, se procedió a generar un gráfico en relación al esfuerzo y los sprints que se tienen para el desarrollo de este trabajo de grado.

Figura 60: Burn-up chart del simulador de tiro con fusil.



Fuente: Elaboración propia, 2020.

3.2.5. Construcción en base de cada característica del simulador para el entrenamiento de tiro con fusiles.

Para la construcción y control de progreso de cada característica se utilizará el método de “parking lot” ejemplificado en la Figura 61.

Figura 61: Método Parking Lot FDD



Fuente: Elaboración propia, 2020.

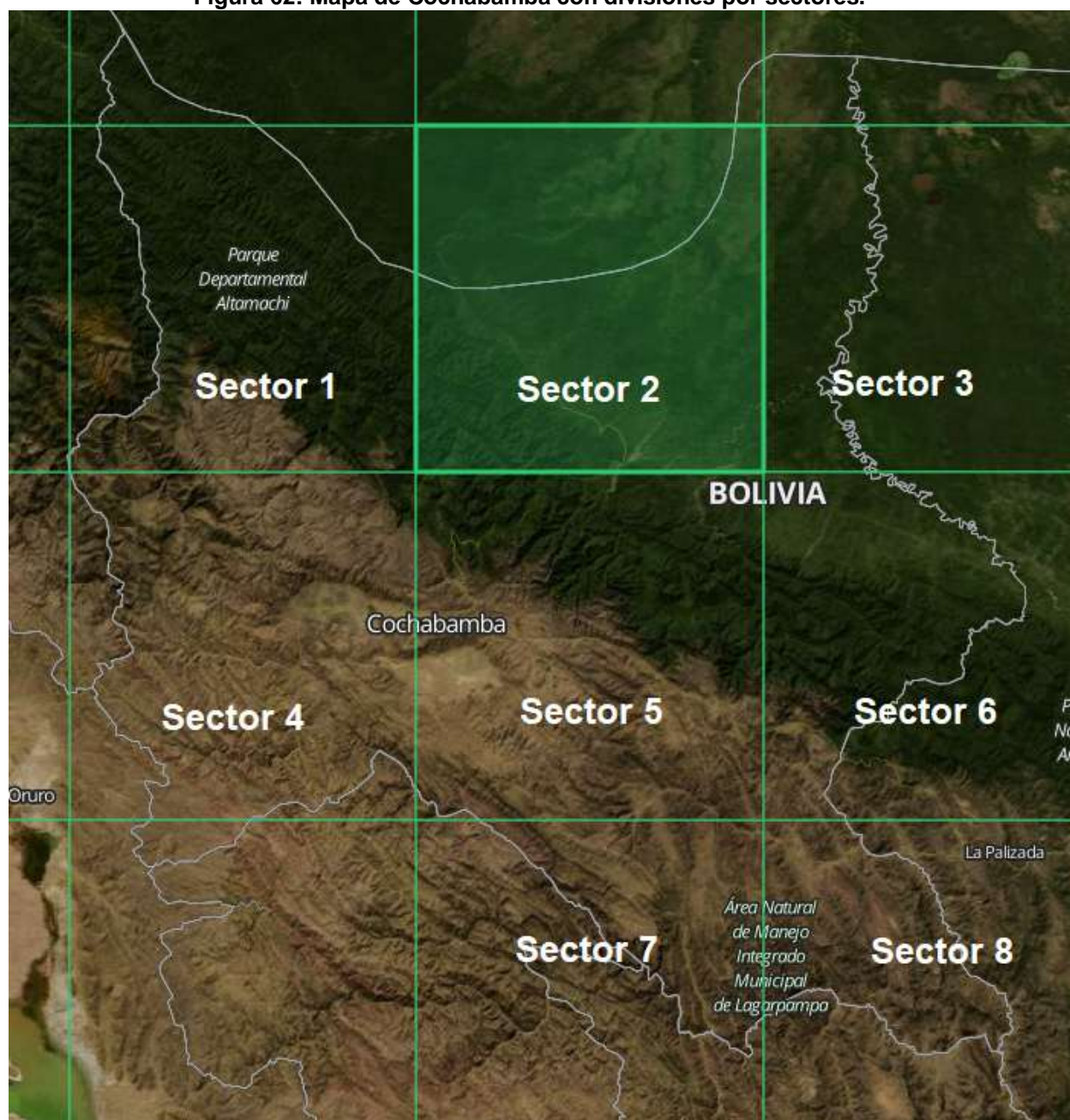
3.3. IMPLEMENTACIÓN DE LOS ENTORNOS DINÁMICOS DE REALIDAD VIRTUAL EN TIEMPO REAL A TRAVÉS DE LOS MÉTODOS PROCEDURALES.

Para la implementación de los entornos dinámicos se utilizará el motor de gráficos UNITY y C# como lenguaje de programación.

3.3.1. Obtención de las curvaturas del entorno mediante el GDEM (Global Digital Elevation Map).

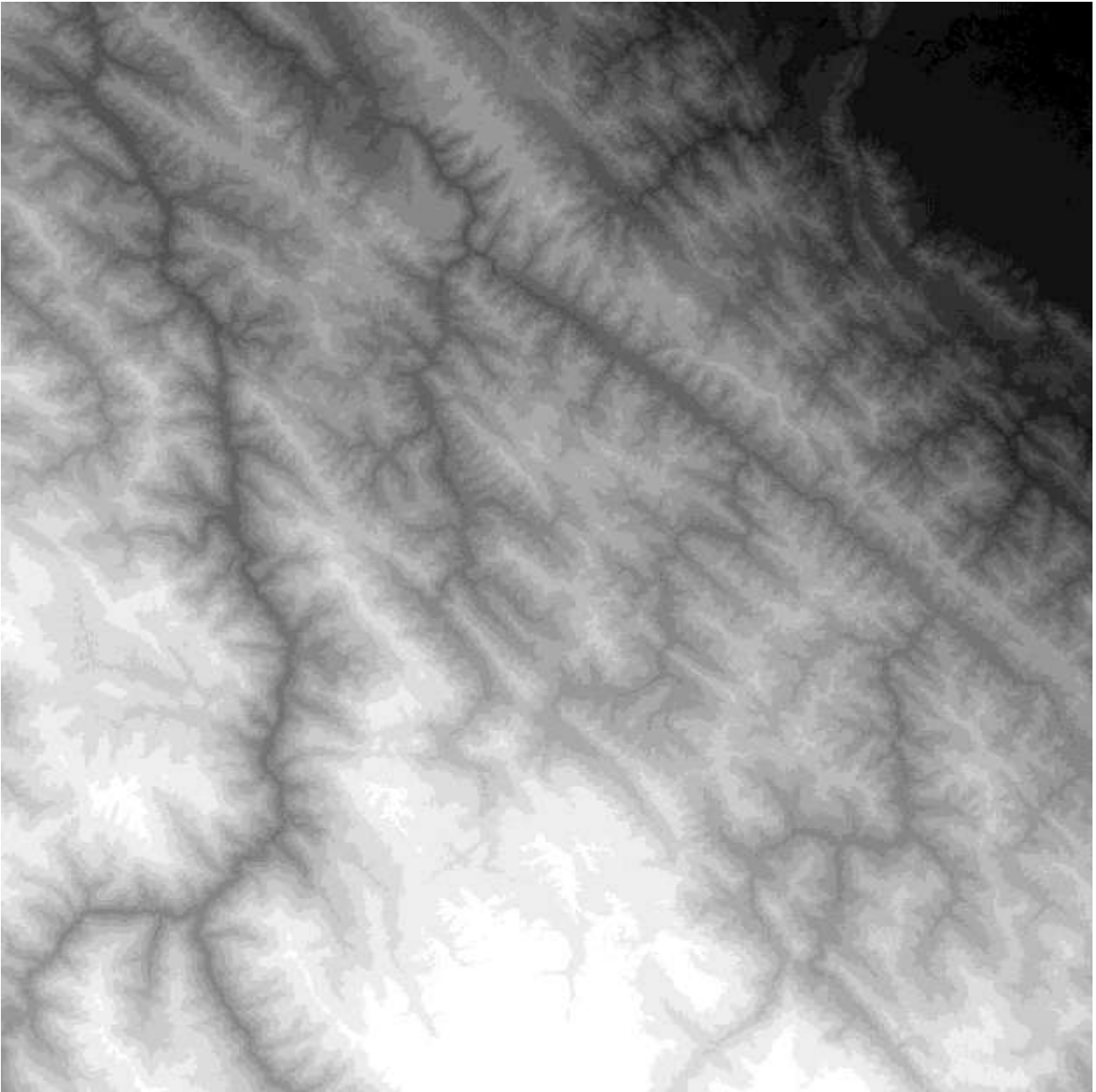
Las curvaturas para el entorno se obtuvieron mediante el ASTER GDEM por un repositorio de la NASA denominado EarthData. Las imágenes comprenden toda Cochabamba y fueron divididas en 8 sectores como se mostrará a continuación.

Figura 62: Mapa de Cochabamba con divisiones por sectores.



Fuente: Elaboración propia, 2020.

Figura 63: Sector 1

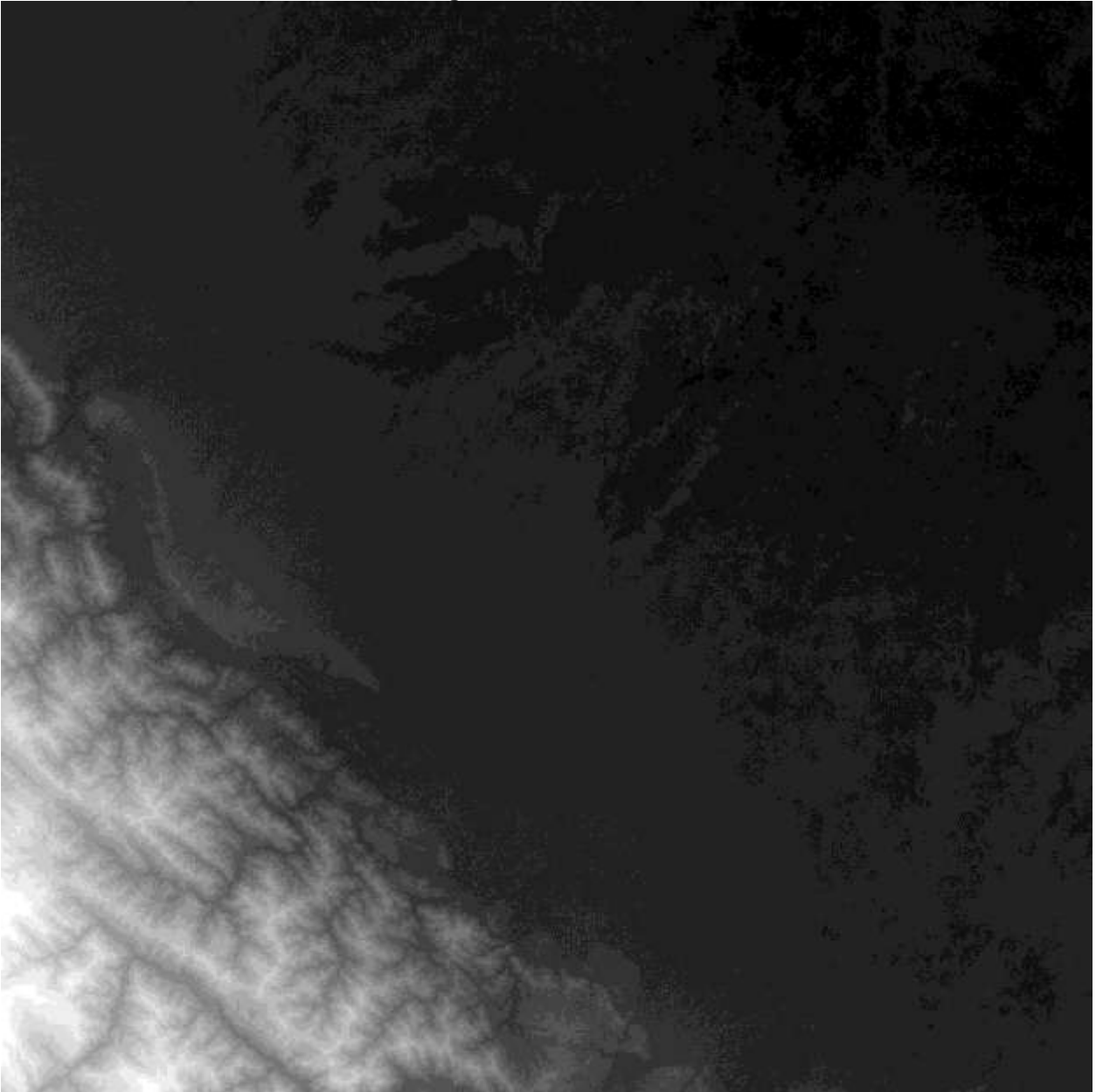


Fuente: Elaboración propia, 2020.

El sector 1 está comprendido dentro de las siguientes coordenadas:

- Coordenada del oeste: -67.0001389
- Coordenada del este: -65.9998611
- Coordenada del norte: -17.9998611
- Coordenada del sur: -19.0001389

Figura 64: Sector 2

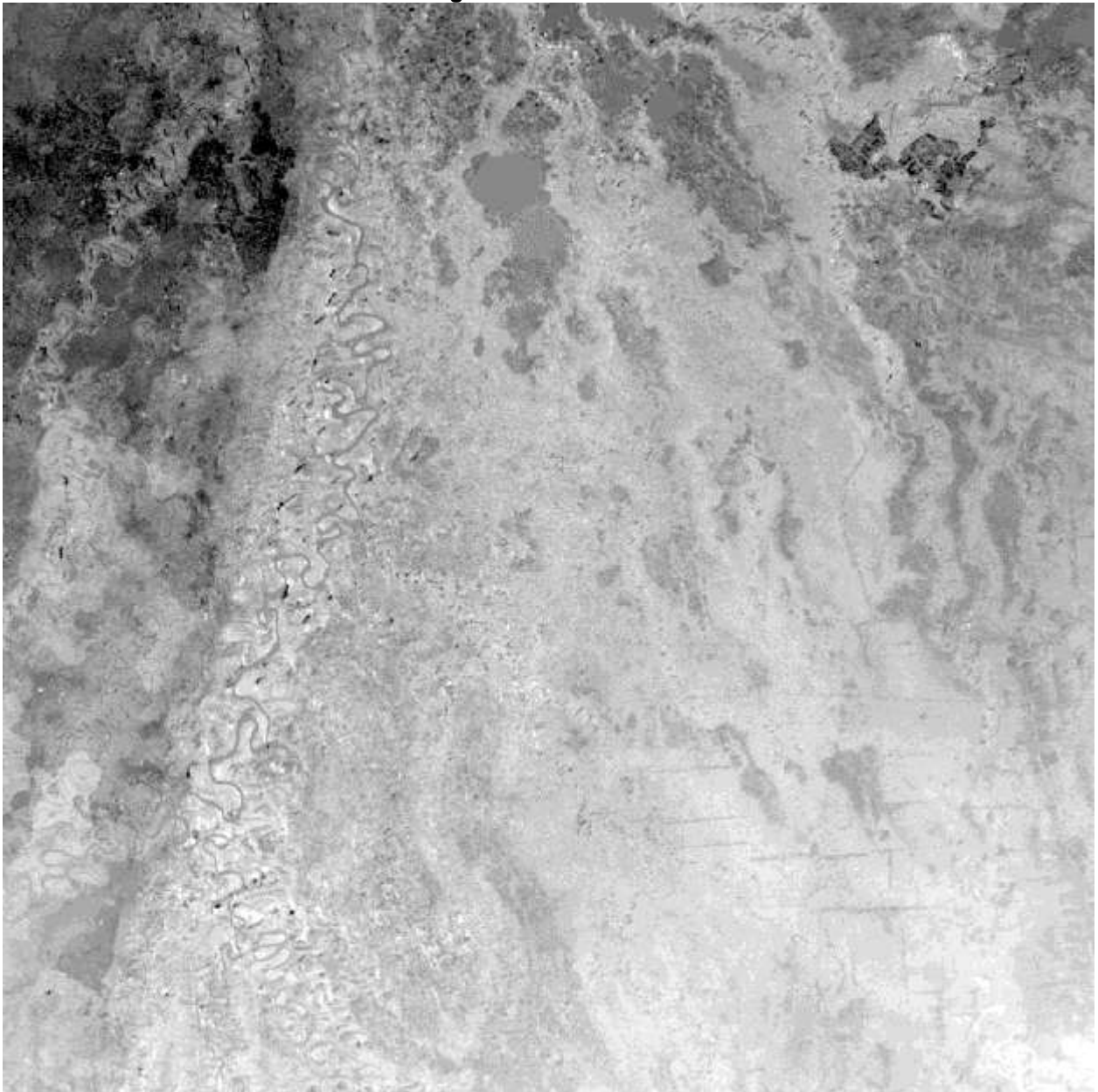


Fuente: Elaboración propia, 2020.

El sector 2 está comprendido dentro de las siguientes coordenadas:

- Coordenada del oeste: -66.0001389
- Coordenada del este: -64.9998611
- Coordenada del norte: -15.9998611
- Coordenada del sur: -17.0001389

Figura 65: Sector 3

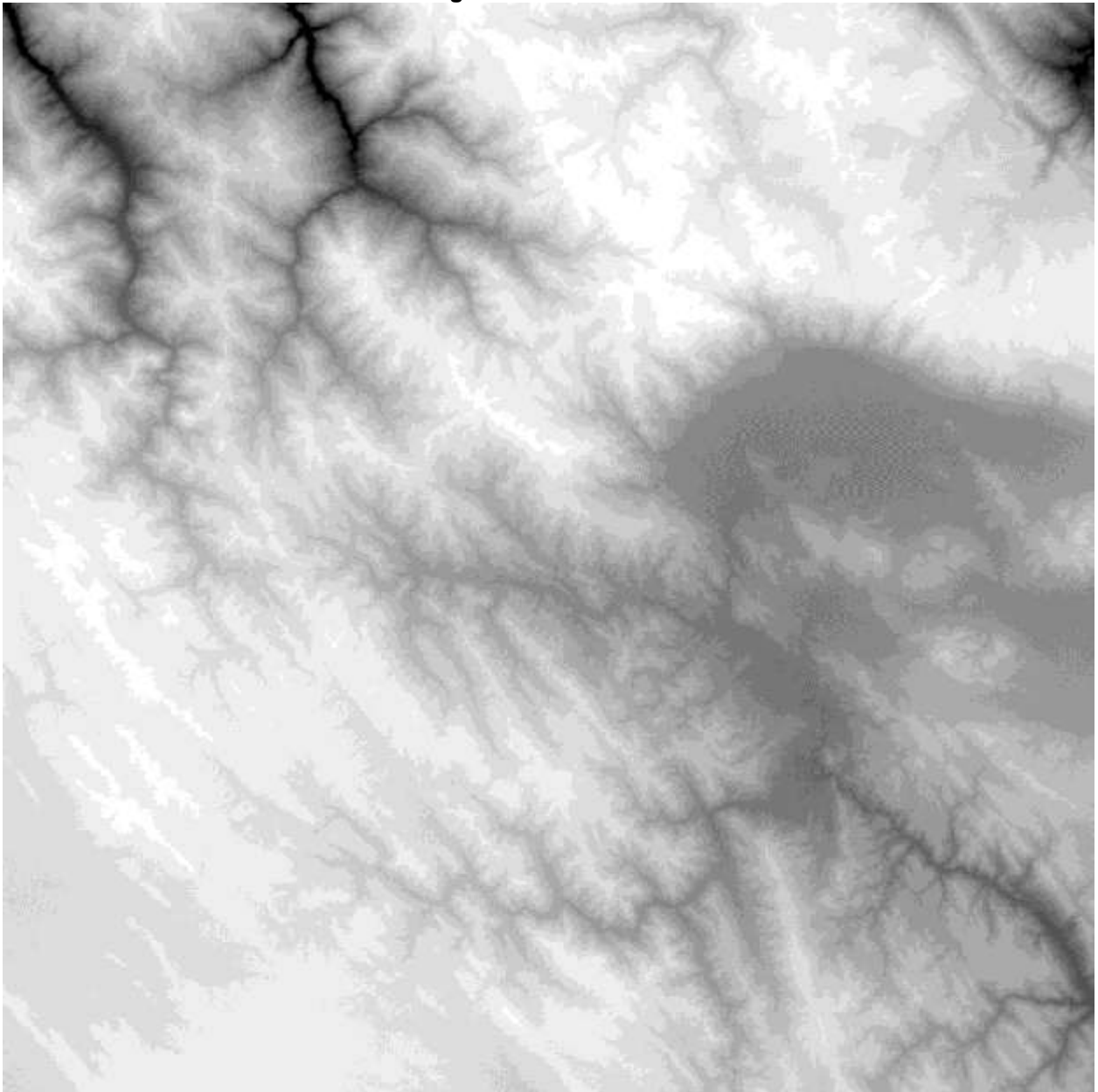


Fuente: Elaboración propia, 2020.

El sector 3 está comprendido dentro de las siguientes coordenadas:

- Coordenada del oeste: -65.0001389
- Coordenada del este: -63.9998611
- Coordenada del norte: -15.9998611
- Coordenada del sur: -17.0001389

Figura 66: Sector 4

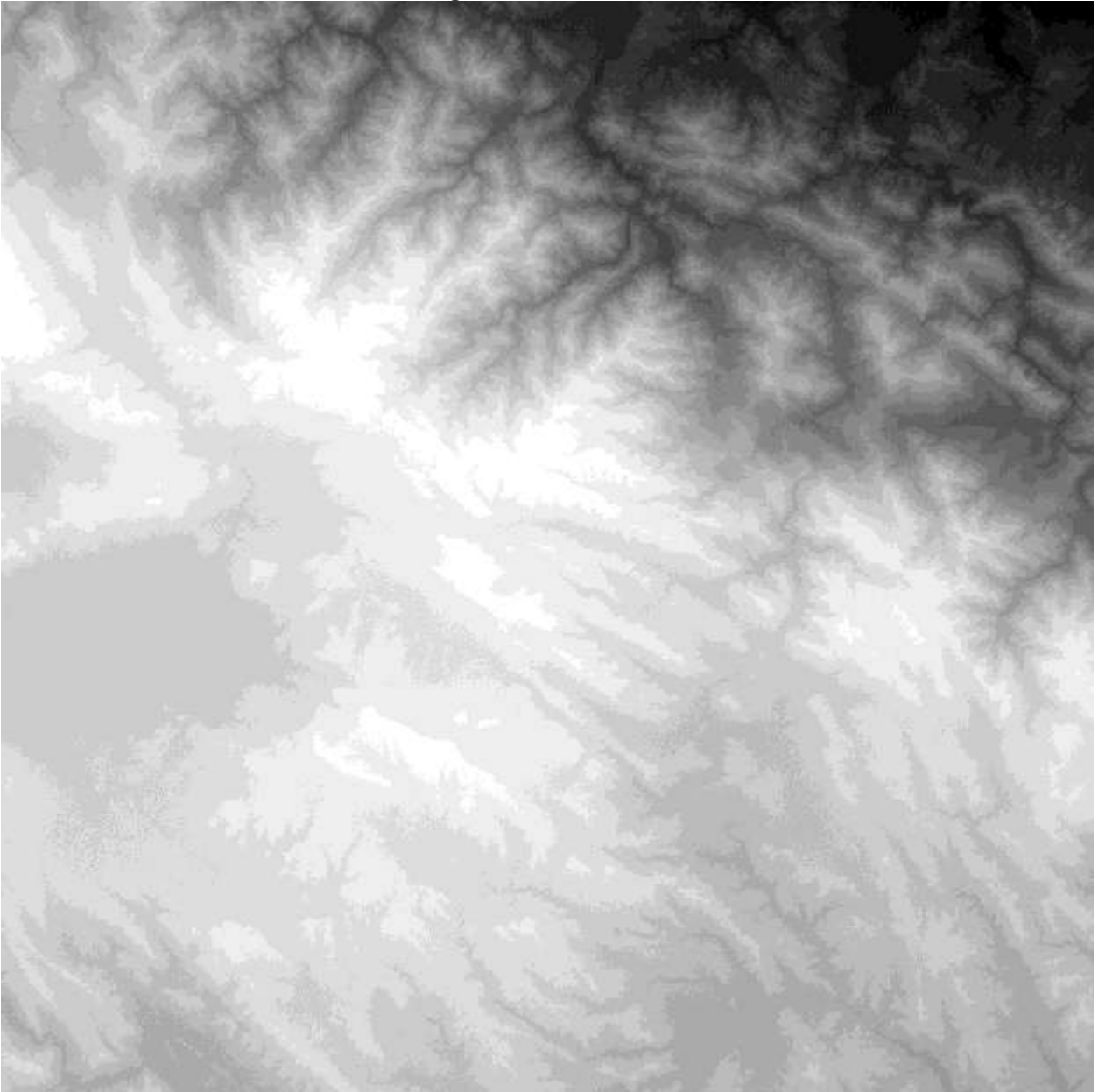


Fuente: Elaboración propia, 2020.

El sector 4 está comprendido dentro de las siguientes coordenadas:

- Coordenada del oeste: -67.0001389
- Coordenada del este: -65.9998611
- Coordenada del norte: -16.9998611
- Coordenada del sur: -18.0001389

Figura 67: Sector 5

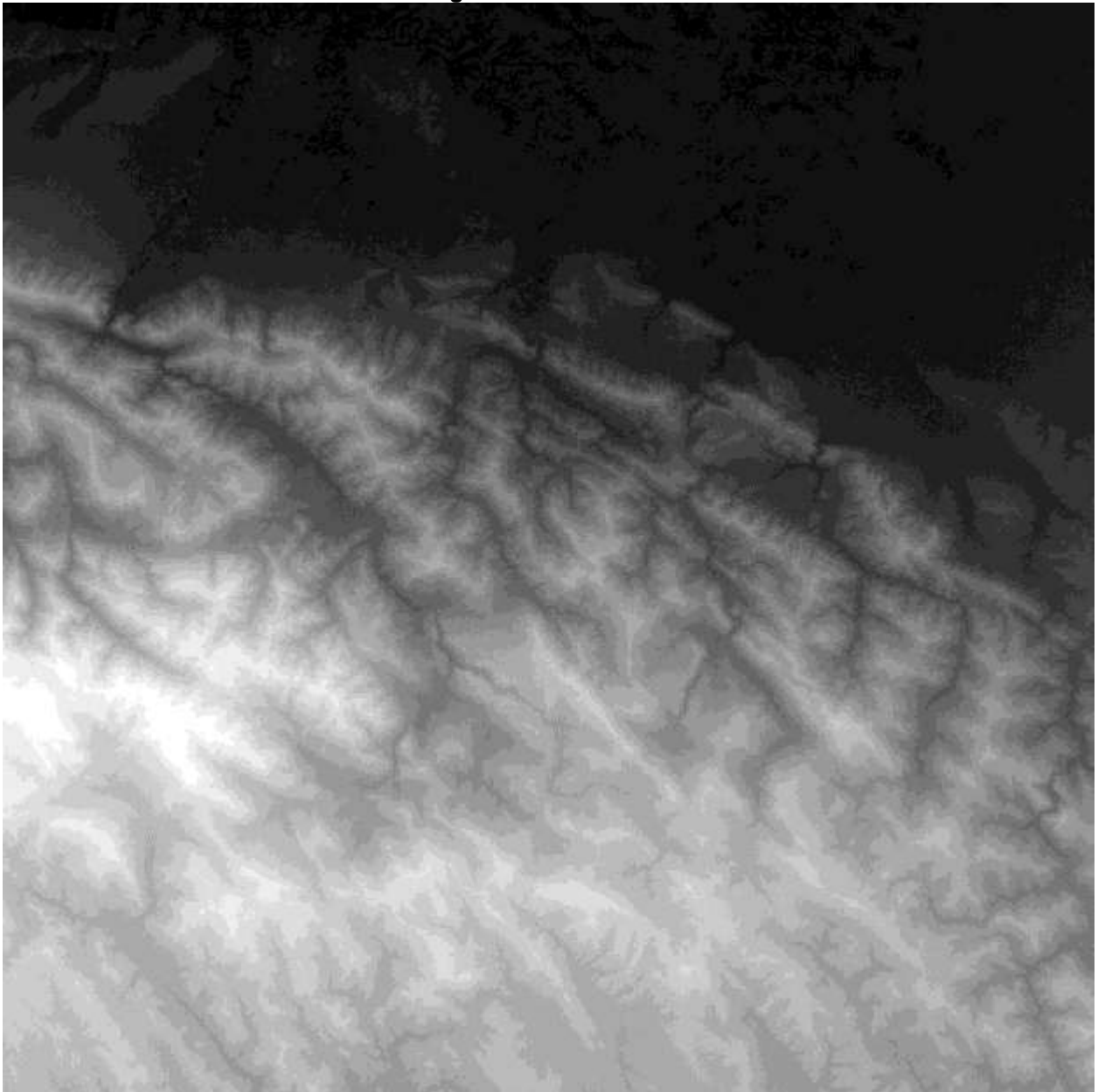


Fuente: Elaboración propia, 2020.

El sector 5 está comprendido dentro de las siguientes coordenadas:

- Coordenada del oeste: -66.0001389
- Coordenada del este: -64.9998611
- Coordenada del norte: -16.9998611
- Coordenada del sur: -18.0001389

Figura 68: Sector 6

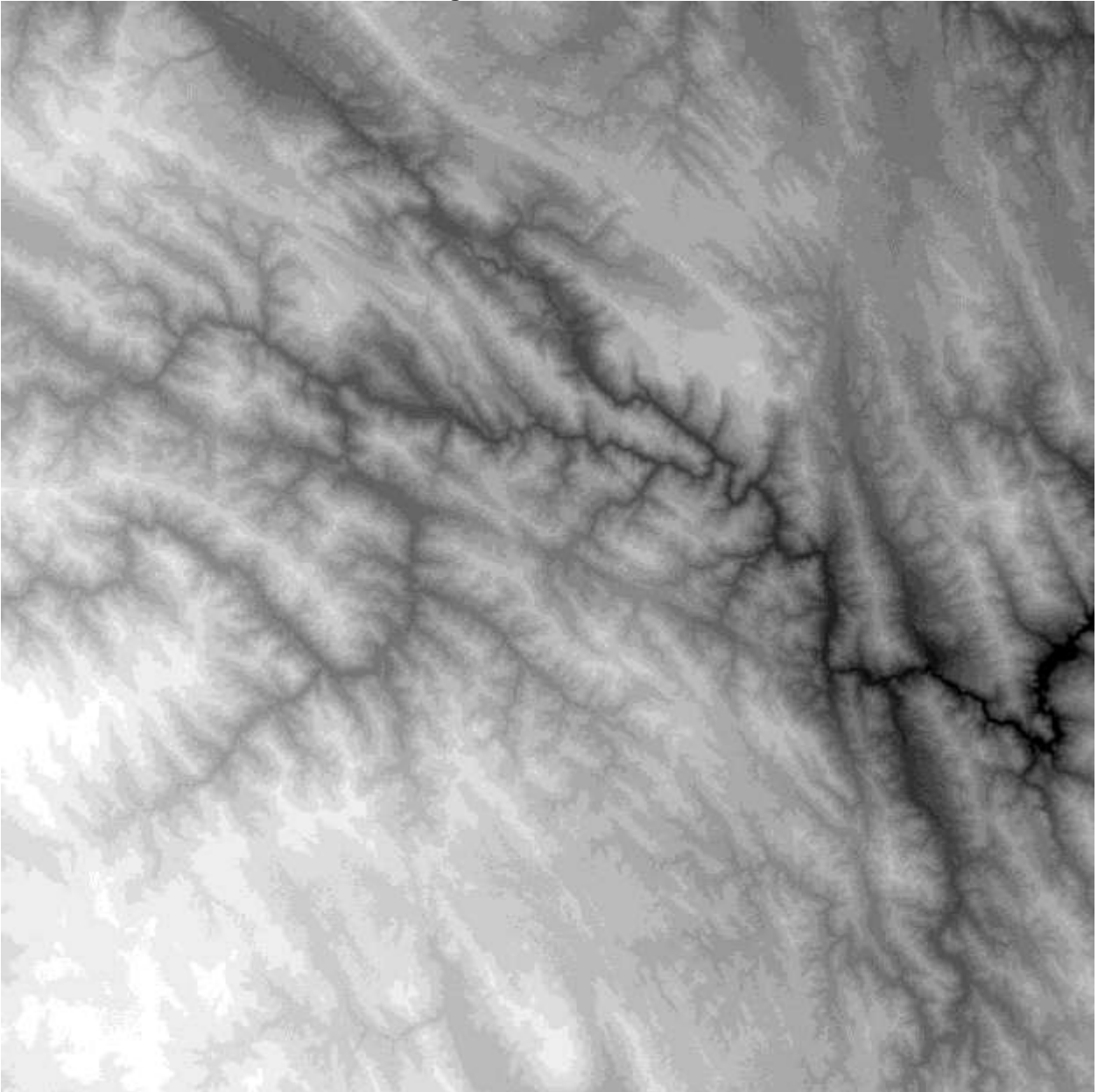


Fuente: Elaboración propia, 2020.

El sector 6 está comprendido dentro de las siguientes coordenadas:

- Coordenada del oeste: -65.0001389
- Coordenada del este: -63.9998611
- Coordenada del norte: -16.9998611
- Coordenada del sur: -18.0001389

Figura 69: Sector 7

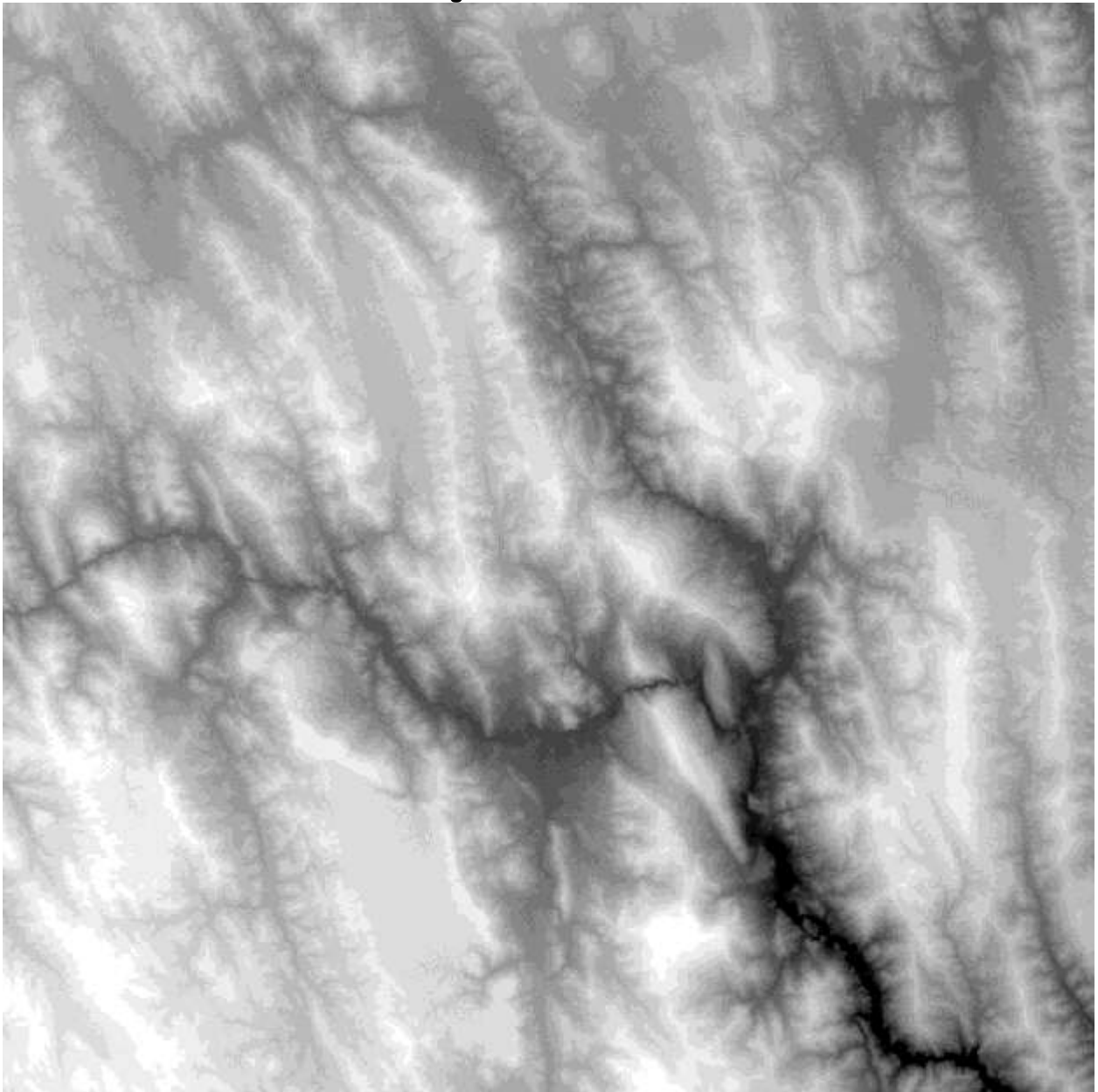


Fuente: Elaboración propia, 2020.

El sector 7 está comprendido dentro de las siguientes coordenadas:

- Coordenada del oeste: -66.0001389
- Coordenada del este: -64.9998611
- Coordenada del norte: -17.9998611
- Coordenada del sur: -19.0001389

Figura 70: Sector 8



Fuente: Elaboración propia, 2020.

El sector 8 está comprendido dentro de las siguientes coordenadas:

- Coordenada del oeste: -65.0001389
- Coordenada del este: -63.9998611
- Coordenada del norte: -17.9998611
- Coordenada del sur: -19.0001389

3.3.2. Codificación del método procedural con las imágenes obtenidas mediante las coordenadas para la generación de terrenos en tiempo real.

Para la codificación del método procedural se utilizó las propiedades de Unity, librerías propias de Unity y la técnica de Nearest Neighbor para el reescalado de imágenes.

3.3.2.1. Métodos de Unity

Los métodos más importantes que fueron utilizados para la generación del terreno a partir de las imágenes fueron los siguientes:

- GetHeights(): Método que permite obtener una matriz con los datos de las alturas del mapa.
- GetPixels(): Este Método nos devuelve una matriz de colores de píxeles , la matriz devuelta es una matriz 2D aplanada.

Los métodos mencionados fueron aplicados de la siguiente manera.

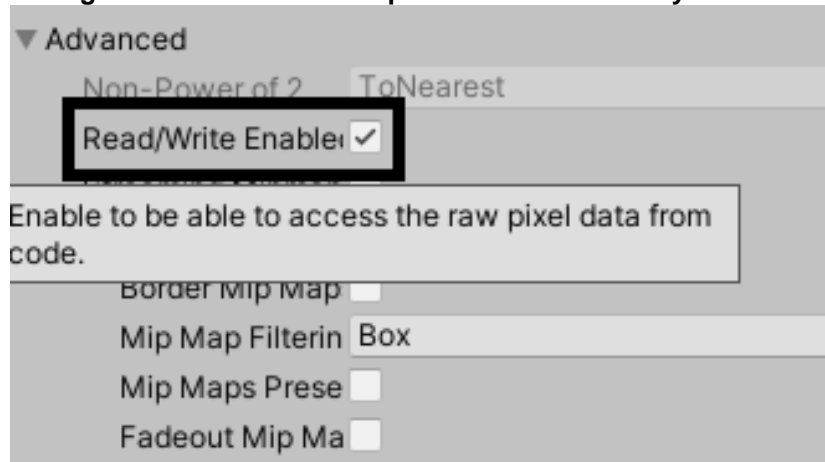
```
1 if (heightmap == null)
2     {
3         EditorUtility.DisplayDialog("Ninguna Textura seleccionada
4 ", "Por favor seleccione una textura.");
5         return;
6     }
7
8     int w = heightmap.width;
9     int h = heightmap.height;
10    int w2 = terrain.heightmapResolution;
11    float[,] heightmapData = terrain.GetHeights(0, 0, w2, w2);
12    Color[] mapColors = heightmap.GetPixels();
13    Color[] map = new Color[w2 * w2];
14    if (w2 != w || h != w)
```

Figura 71: Métodos de Unity C#

Fuente: Elaboración propia, 2020.

En el cuadro anterior se encuentra el mapeo realizado a la imagen obteniendo el alto y ancho de la imagen del mapa, para que esto pueda pasar por el script hay que habilitar los permisos como se mostrara a continuación.

Figura 72: Habilitación de permisos de escritura y lectura.



Fuente: Elaboración propia, 2020.

3.3.2.2. Nearest Neighbor

En caso de pasar una imagen no simétrica, ejemplo 1024 x 1028 se aplica la técnica del Nearest Neighbor la cual permite realizar un reescalado eficiente, esta técnica está basada en la interpolación lineal.

Basándonos en el pseudocódigo realizado en Python, lo pasamos al lenguaje nativo de Unity el cual es C#.

```

1  sourceX = int( round ( targetX / targetWidth * sourceWidth ) )
2  sourceY = int( round ( targetY / targetHeight * sourceHeight ) )
3
4  def nearestNeighborScaling( source, newWid, newHt ):
5      target = makeEmptyPicture(newWid, newHt)
6      width = getWidth( source )
7      height = getHeight( source )
8      for x in range(0, newWid):
9          for y in range(0, newHt):
10             srcX = int( round( float(x) / float(newWid) * float(width) ) )
11             srcY = int( round( float(y) / float(newHt) * float(height) ) )
12             srcX = min( srcX, width-1)
13             srcY = min( srcY, height-1)
14             tarPix = getPixel(target, x, y )
15             srcColor = getColor( getPixel(source, srcX, srcY) )
16             setColor( tarPix, srcColor)
17
18     return target

```

Figura 73: Nearest Neighbor Python

Fuente: Elaboración propia, 2020.

Y el implementado de nuestro script de Unity es el siguiente:

```

1 if (w2 != w || h != w)
2     {
3
4         if (heightmap.filterMode == FilterMode.Point)
5         {
6             float dx = (float)w / (float)w2;
7             float dy = (float)h / (float)w2;
8             for (int y = 0; y < w2; y++)
9             {
10                 if (y % 20 == 0)
11                 {
12                     EditorUtility.DisplayProgressBar("Resize",
13 "Calculating texture", Mathf.InverseLerp(0.0f, w2, y));
14                 }
15                 int thisY = Mathf.FloorToInt(dy * y) * w;
16                 int yw = y * w2;
17                 for (int x = 0; x < w2; x++)
18                 {
19                     map[yw + x] =mapColors[Mathf.FloorToInt(thisY
20 + dx * x)];
21                 }
22             }
23         }

```

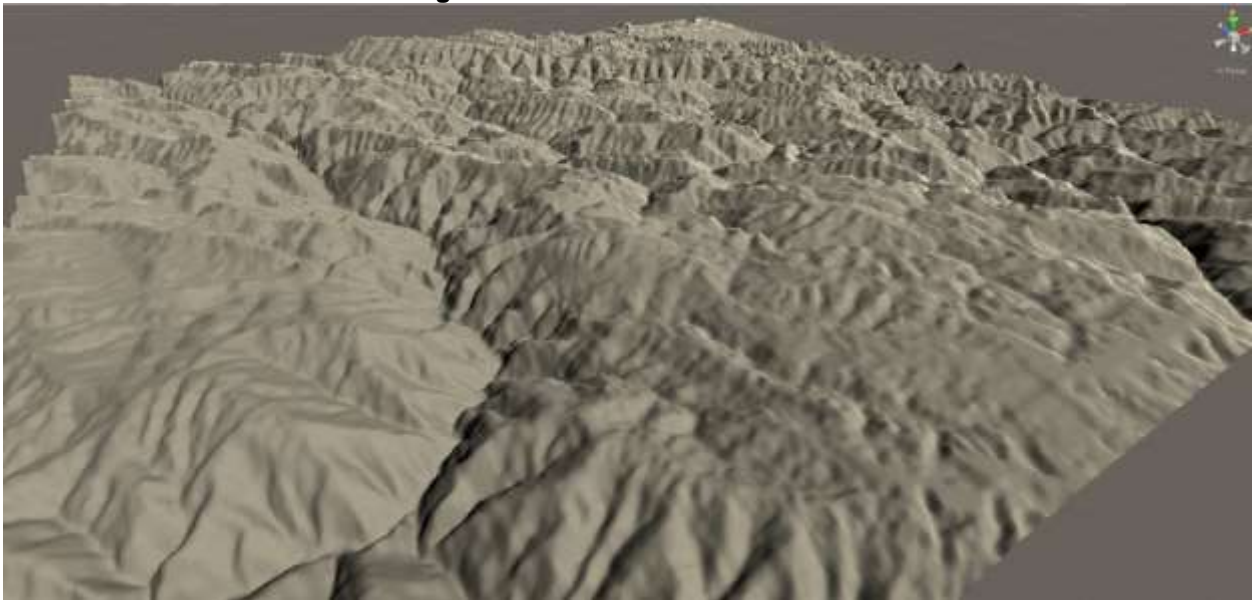
Figura 74: Nearest Neighbor C#

Fuente: Elaboración propia, 2020.

3.3.3. Verificación del funcionamiento del método procedural y su funcionalidad a la hora de generar el terreno en tiempo real.

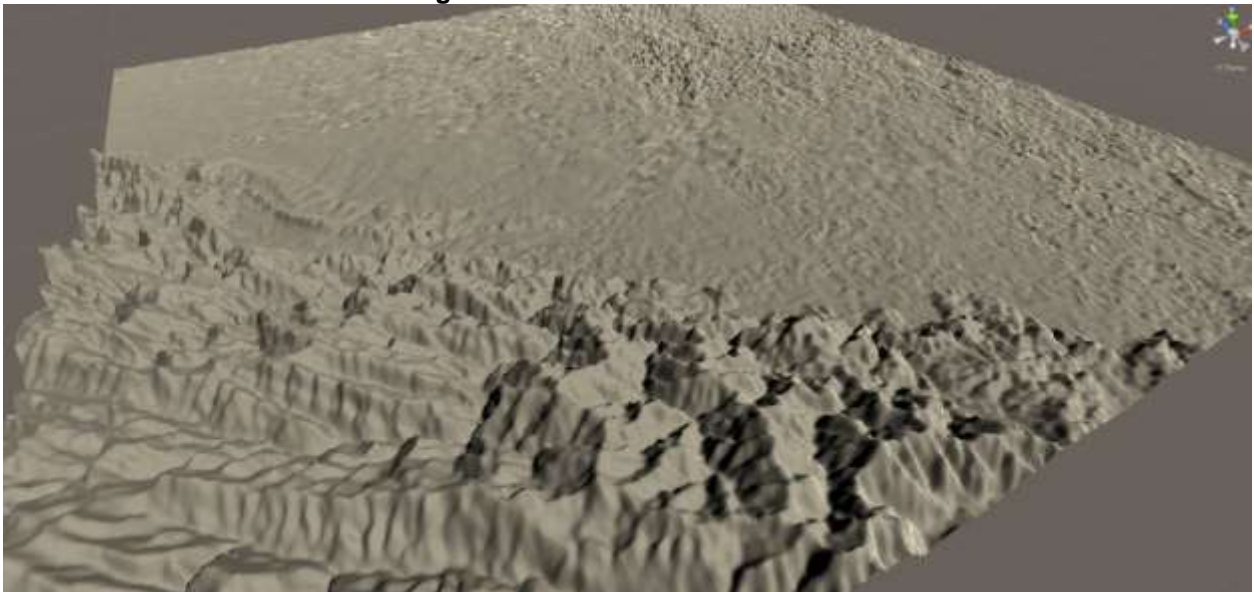
Se ejecuto el script con cada una de las imágenes sectorizadas para probar su funcionamiento, el cual tuvo el siguiente resultado.

Figura 75: Sector 1 renderizado



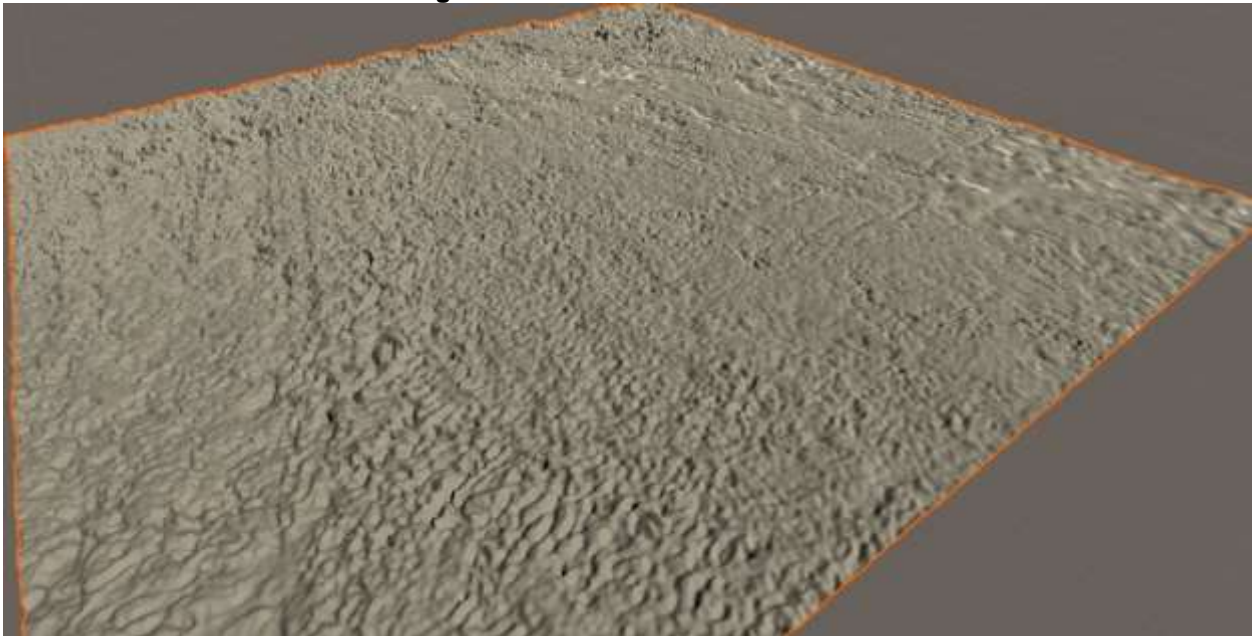
Fuente: Elaboración propia, 2020.

Figura 76: Sector 2 renderizado



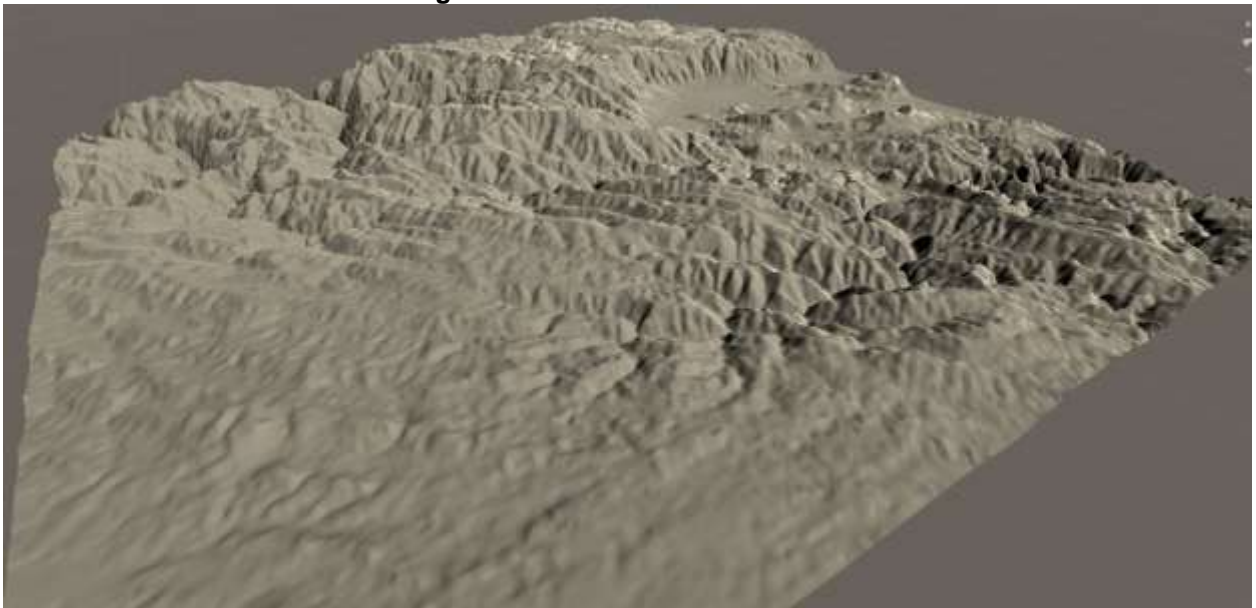
Fuente: Elaboración propia, 2020.

Figura 77: Sector 3 renderizado



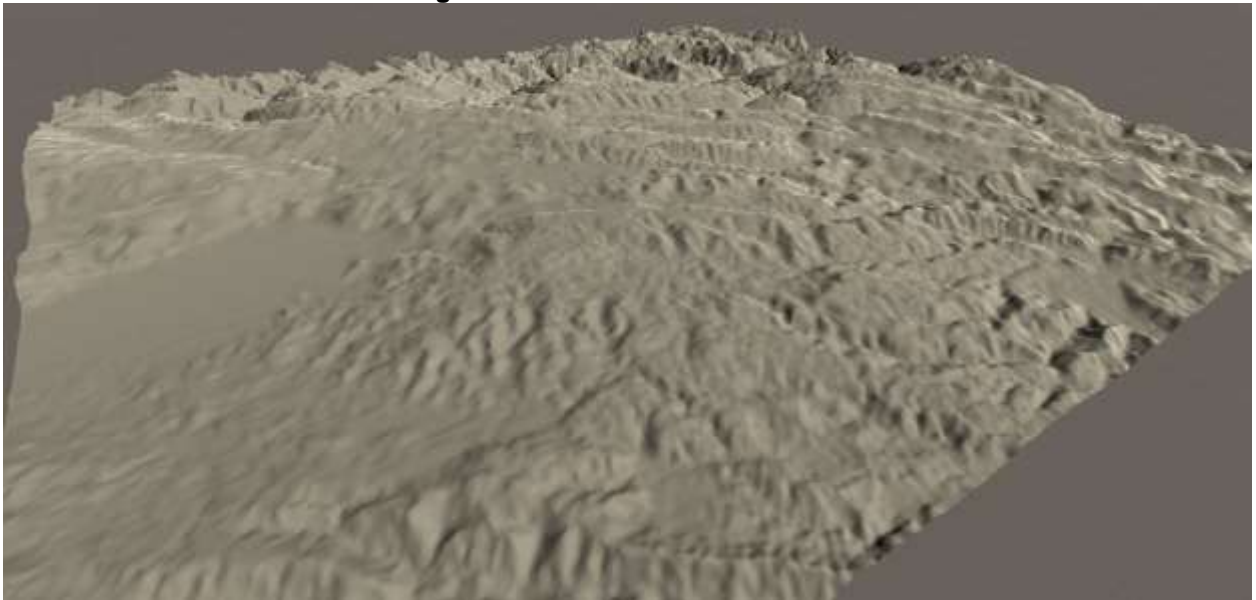
Fuente: Elaboración propia, 2020.

Figura 78: Sector 4 renderizado



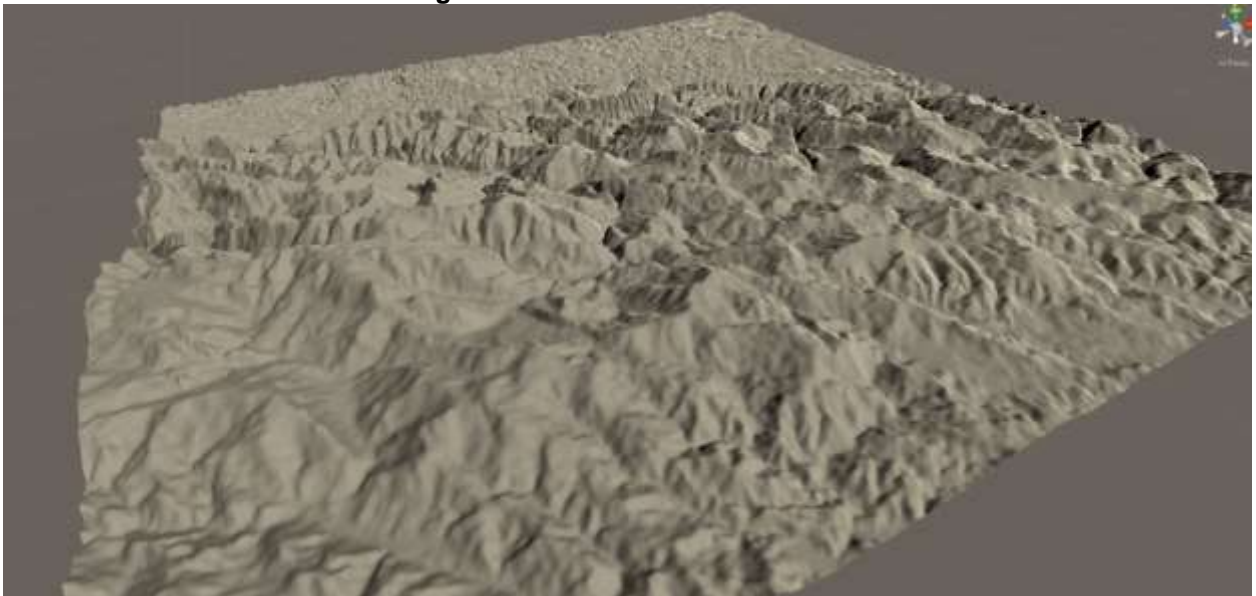
Fuente: Elaboración propia, 2020.

Figura 79: Sector 5 renderizado



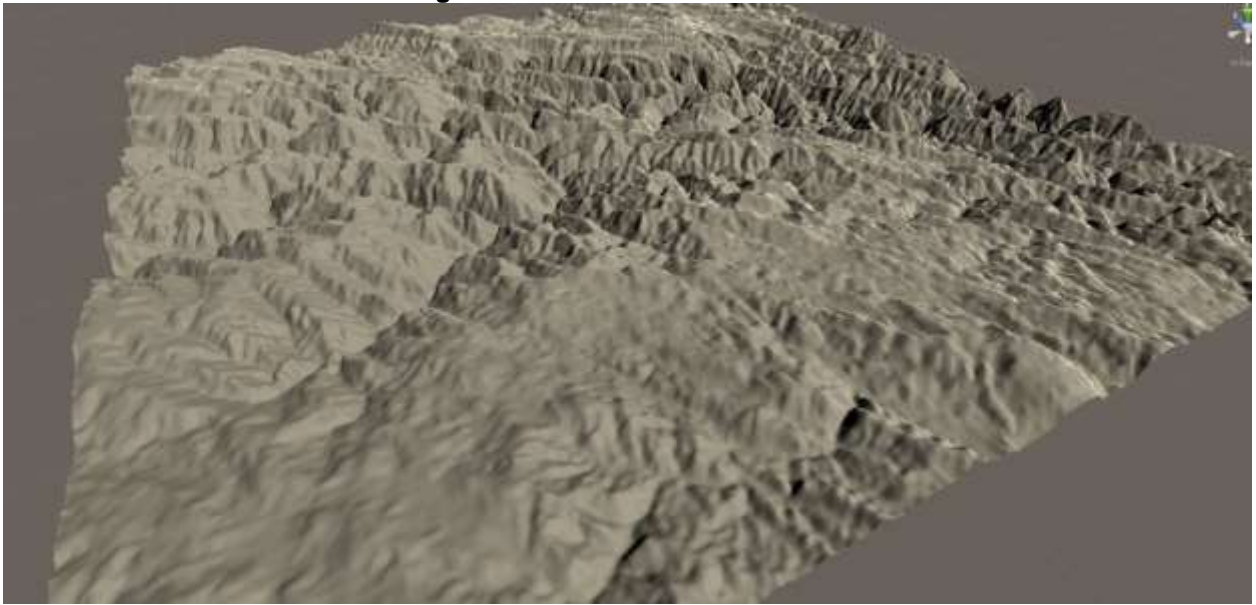
Fuente: Elaboración propia, 2020.

Figura 80: Sector 6 renderizado



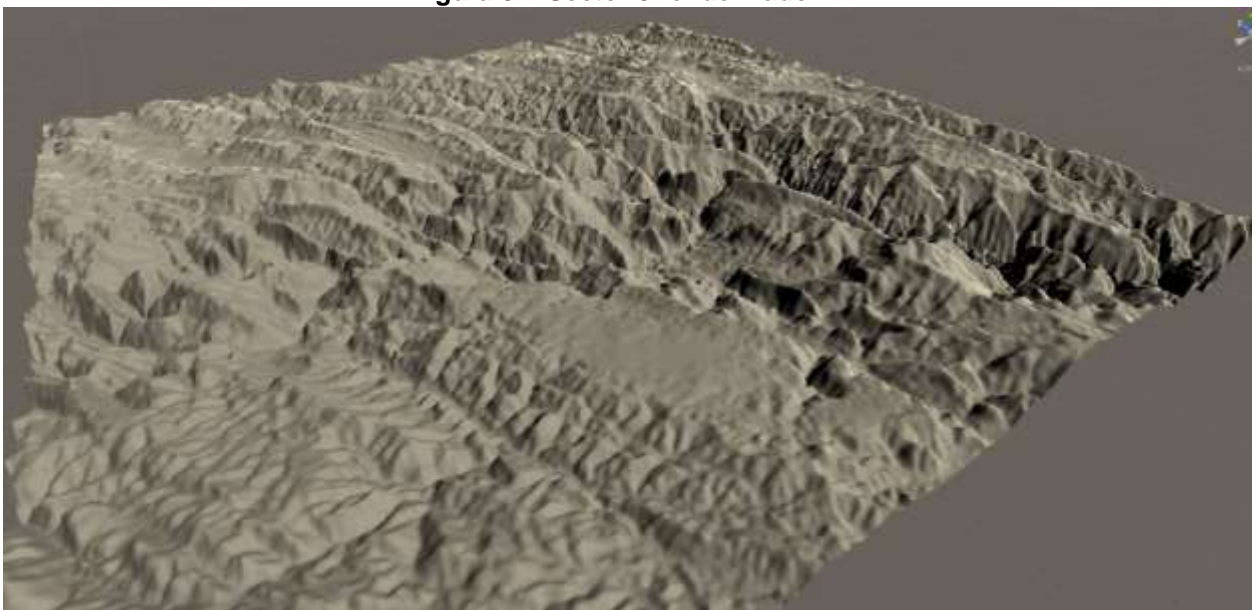
Fuente: Elaboración propia, 2020.

Figura 81: Sector 7 renderizado



Fuente: Elaboración propia, 2020.

Figura 82: Sector 8 renderizado



Fuente: Elaboración propia, 2020.

Llegamos a la conclusión de que el script está funcionando correctamente, y está renderizando en tiempo real sin ningún problema.

3.4. ESTABLECIENDO LAS CARACTERÍSTICAS DEL DISPARO EN LOS CLIMAS DIVERSOS DE ACUERDO A LOS TERRENOS DINÁMICOS.

Para establecer las características del disparo en los climas diversos de acuerdo a los terrenos dinámicos, se procederá a recolectar información sobre las diferentes variables climatológicas, para que después se puedan implementar como indican las acciones de este objetivo específico.

3.4.1. Recopilación de información de las variables climatológicas.

Para la recolección de información de las variables climatológicas, tomaremos en cuenta los siguientes puntos, los cuales nos permitirá tener una idea de las variables climatológicas donde desarrollaremos el entorno del simulador.

Según la teoría de balística se debería tomar en cuenta las siguientes variables:

- Altitud.
- Temperatura.
- Viento.

Altitud. – Para la toma de altitud se utilizó el coprocesador M11 del iPhone X con el cual se obtuvo lo siguiente:

Figura 83: Altitud obtenida con el coprocesador M11



Fuente: Elaboración propia, 2020.

Como se puede observar, en la Figura 83, nos muestra que nos encontramos a una elevación de 8510 pies, para llevar esta medida a metros se realizara el siguiente calculo:

$$8510ft * \left(\frac{1m}{3.281ft} \right) = 2593.7214 m$$

ft = pies

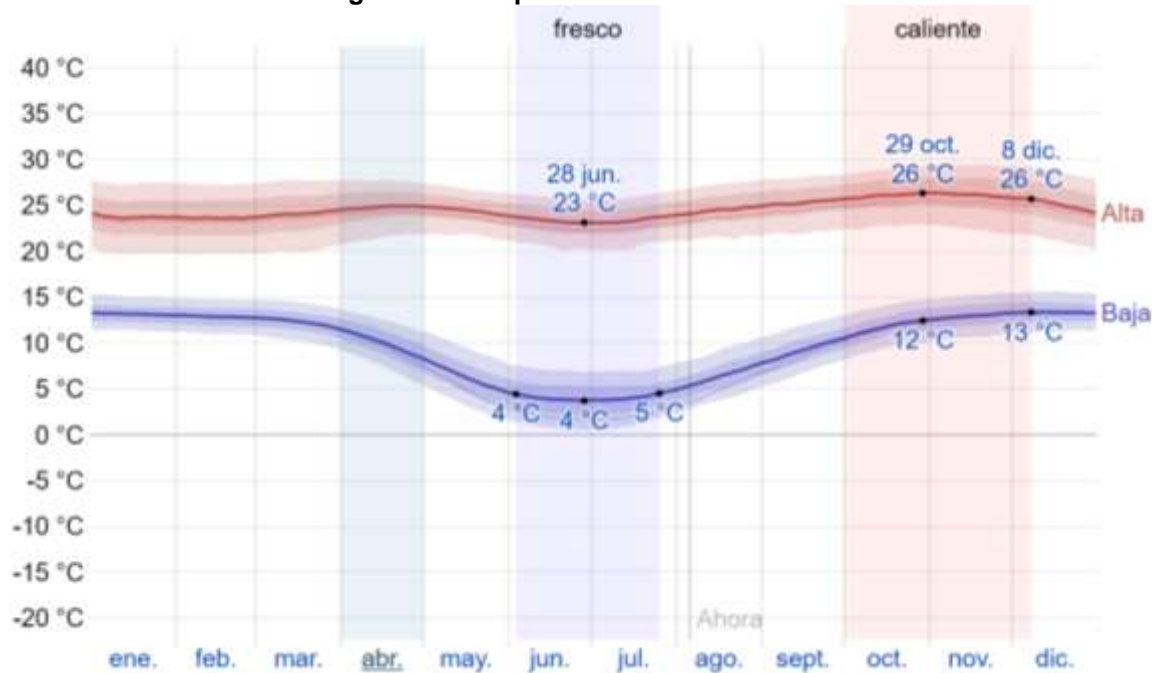
m = metros

El resultado obtenido con el coprocesador M11 fue de 2593 m.s.n.m.

Temperatura. – Para obtener los datos de la temperatura tomaremos como referencia los datos de weatherspark el cual tiene una base de datos enorme con respecto a las diferentes variables climatológicas de muchos países y ciudades.

Los datos de Cochabamba en cuanto a temperatura indican lo siguiente:

Figura 84: Temperatura de Cochabamba



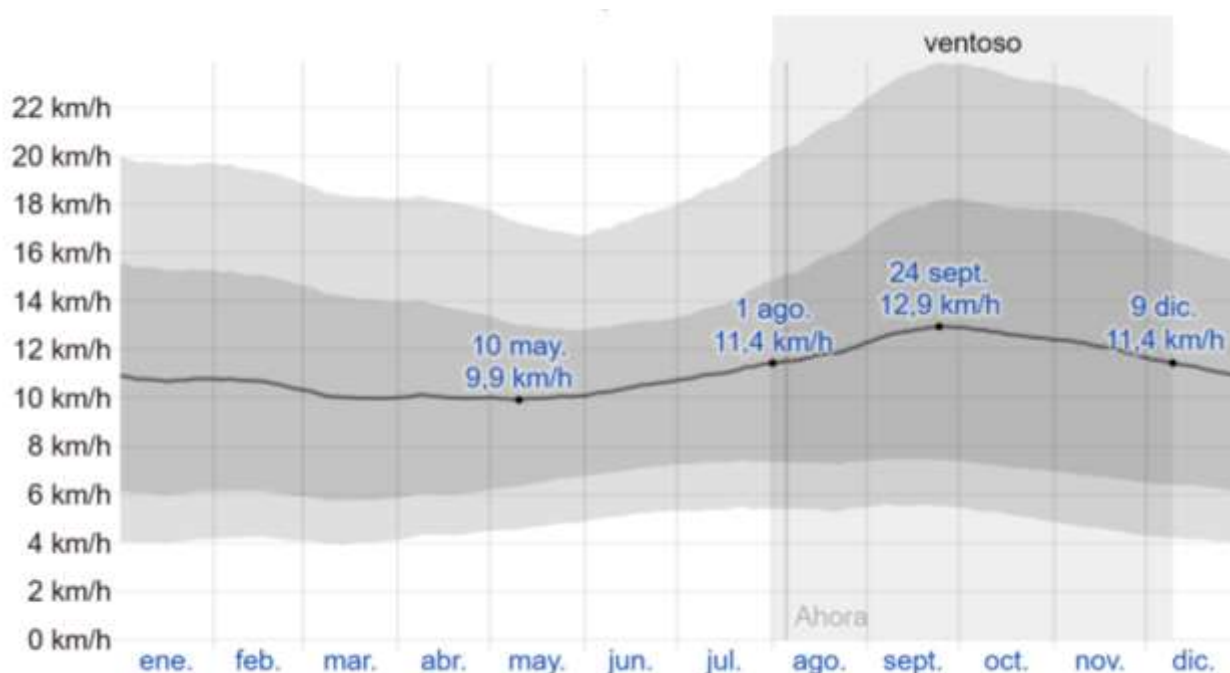
Fuente: Weather Spark, 2020.

Como podemos observar en la Figura 84, en el departamento de Cochabamba se tiene una temperatura máxima promedio de entre 23 °C y 26 °C en el transcurso de todo el año, y en cuanto a la temperatura mínima se tiene un promedio de entre 4 °C y 13 °C esta información permitirá establecernos un rango de parámetros en la implementación del clima del entorno.

Viento. – Para los datos del viento tomaremos como referencia los datos provistos por Weather Spark. Estos datos del viento nos permitirán agregar una dirección a la

trayectoria de la munición, debido a que esta podrá hacer variar la trayectoria del proyectil en base a la dirección de la corriente del viento. En la Figura 85 se mostrarán los datos del viento los cuales están considerados a 10 metros sobre el suelo.

Figura 85: Velocidad del viento en Cochabamba



Fuente: Weather Spark, 2020.

Como podemos observar en la gráfica durante el año en Cercado, Cochabamba el promedio de velocidad del viento esta entre unos 9.9 km/h y 11.4 km/h, en cuanto a la velocidad mínima que se alcanza durante el año, se tiene un valor de entre 4 km/h y 6 km/h. Para la velocidad máxima promedio que se alcanza durante el año se tiene entre 18km/h y 22 km/h.

Se tendrá que tomar en cuenta un factor extra del viento el cual veremos en la Figura 85 y Figura 86.

Figura 86: Dirección del viento en Cochabamba



Fuente: Weather Spark, 2020.

Como se puede observar en la Figura 86, la predominancia de la dirección del viento en Cercado, Cochabamba es con dirección al norte, como segundo al oeste, como tercero al este y como último al sur, pero es casi nulo su porcentaje.

3.4.2. Codificación de los atributos del clima relacionado al entorno.

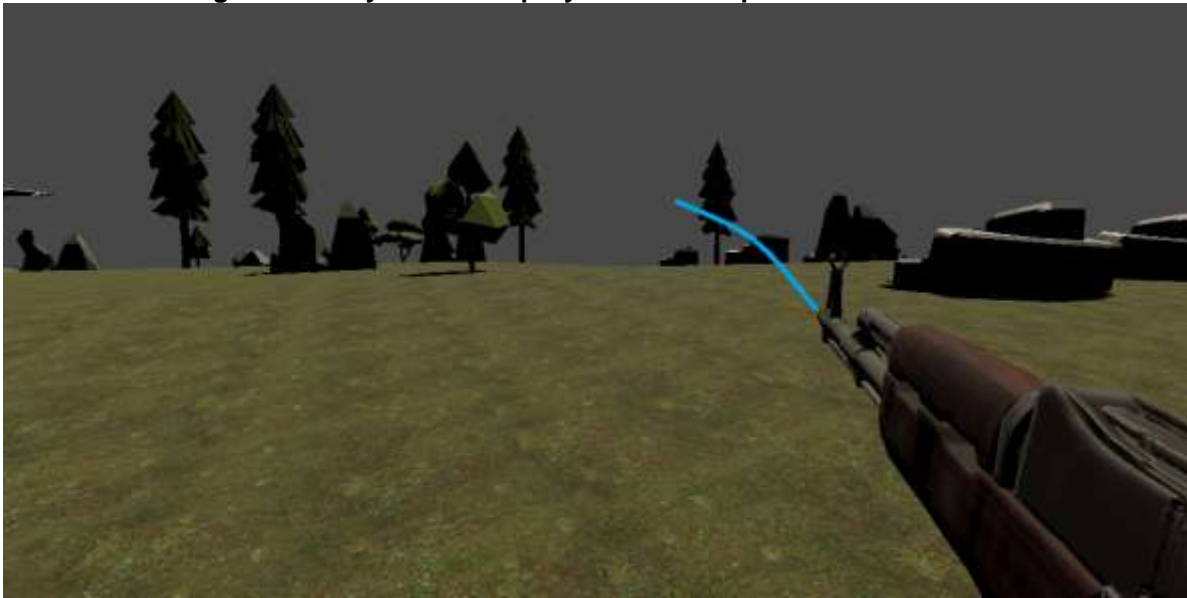
Para la codificación de los atributos se tomará en cuenta las variables climatológicas mencionadas en el anterior punto. Ver Anexo F.

El script que se realizara tiene como base principal el uso de la librería Vector 3, la cual incluye funciones físicas, y también métodos útiles para el trabajo con fuerzas externas.

Para la implementación se tomó en cuenta un rango de parámetros, en este caso desde 9.9 km/h hasta 22 km/h, siendo esta una variable denominada “vientoForce” la cual contendrá los valores del viento cuando el simulador sea ejecutado. Las variables de clima son modificables antes de su ejecución, y no son variables cuando el simulador ya está en ejecución.

La salida que tendremos en las variaciones del viento, afectará la trayectoria del proyectil como se verá a continuación.

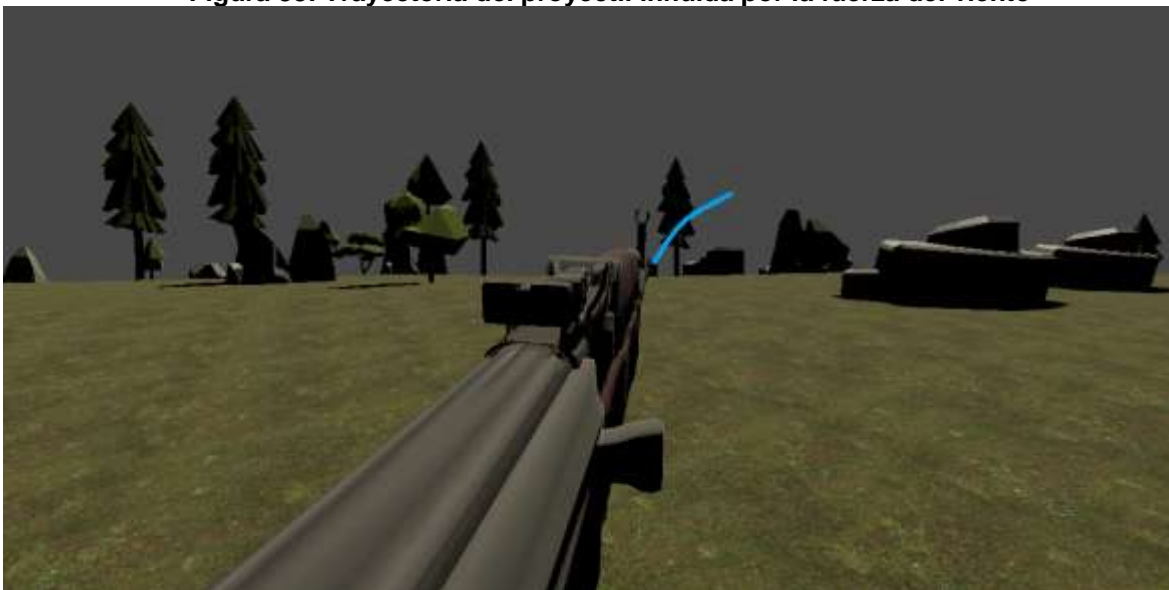
Figura 87: Trayectoria del proyectil influida por la fuerza del viento



Fuente: Elaboración propia, 2020.

Como se puede ver en la Figura 87, existe una variación en la trayectoria del proyectil, en este caso se tomó una velocidad del viento de 18 km/h proveniente del Oeste (eje X negativo).

Figura 88: Trayectoria del proyectil influida por la fuerza del viento



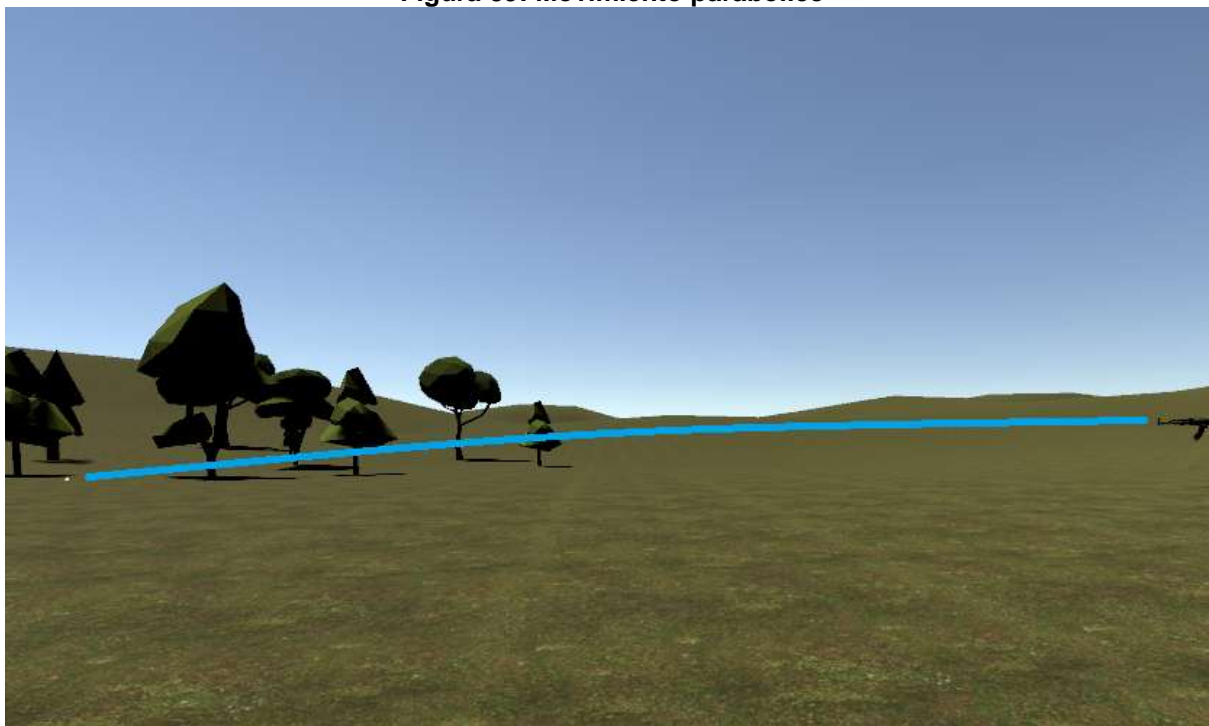
Fuente: Elaboración propia, 2020.

En la Figura 88 se muestra de igual forma solo que con el viento proveniente desde el Este a una velocidad de 20km/h.

3.4.3. Codificación de la física relacionada a la trayectoria de disparo.

Para la codificación de la física relacionada a la trayectoria del disparo utilizaremos las fórmulas de física mencionadas en el marco teórico, tomando en cuenta algunos valores de los atributos del clima relacionado al entorno.

Figura 89: Movimiento parabólico



Fuente: Elaboración propia, 2020.

En la Figura 89 se puede apreciar el movimiento parabólico efectuado en el proyectil, en el que fue aplicada las fórmulas físicas de movimiento parabólico. Ver Anexo F para ver de forma más detallada el script del arma en el cual se instancia los parámetros de la física.

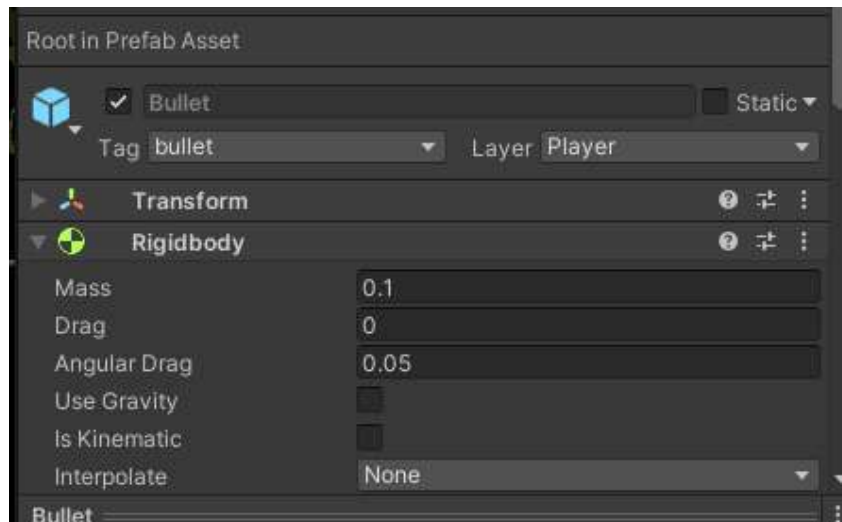
Figura 90: Propiedades del fusil de asalto.



Fuente: Elaboración propia, 2020.

El Rigidbody de Unity añade propiedades físicas a los GameObjects del mismo el cual se pueden acceder por código.

Figura 91: Propiedades del proyectil.



Fuente: Elaboración propia, 2020.

De igual forma se añade el Rigidbody al objeto del proyectil, esto para que se le aplique la gravedad y las propiedades físicas que se tienen en el código.

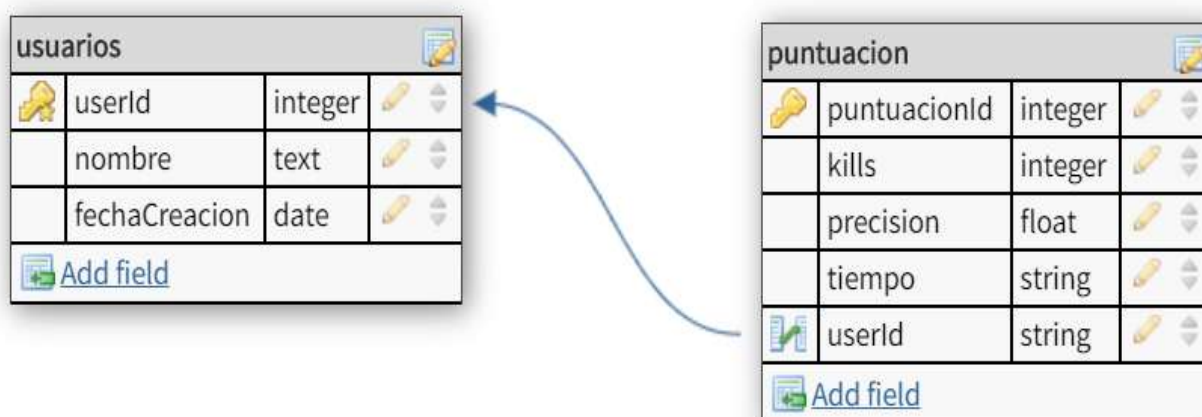
3.5. IMPLEMENTACIÓN DE UNA TABLA ESTADISTICA DE TIRO PARA OBSERVAR LA EVOLUCIÓN DEL USUARIO.

Para la implementación de la base de datos utilizaremos SQLite y para la parte visual se utilizará el Canvas y los diferentes Game Objects de UI que vienen incluidos en Unity para el desarrollo de interfaces. A continuación, se procederá a desglosar los siguientes puntos con respecto a este objetivo. Para ver los scripts de manera detallada ver Anexo G.

3.5.1. Diseño de la base de datos (SQL) para el almacenamiento de datos estadísticos por usuario.

Para el diseño de esta base de datos se tomara en cuenta lo siguiente, una tabla para usuarios el cual contendrá los campos de su id, nombre del usuario que será único y el tiempo de creación, y para la otra tabla se tendrán los campos de la id de puntuación(puntuacionId), los kills(aciertos), precisión y tiempo, también se tendrá una clave foránea por parte de esta para relacionar con la tabla de usuarios, el diseño quedara de la siguiente manera.

Figura 92: Diseño de la DB



Fuente: Elaboración propia, 2020.

Dado el diseño se pasará a realizar la implementación de la misma dentro de lo que es SQLite.

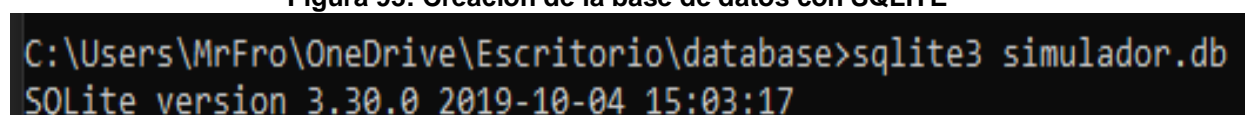
3.5.2. Creación de la base de datos (SQL)

Para la creación de la base de datos con SQLite se procederá de la siguiente manera. Inicialmente nos dirigiremos al path donde se quiere crear la base de datos, utilizando el comando `cd` en CMD de Windows. Y simplemente se escribirá el comando.

>sqlite3 “nombre de la base de datos”.db

Y se creará un archivo de tipo db como se verá en la Figura 93.

Figura 93: Creación de la base de datos con SQLITE




```
C:\Users\MrFro\OneDrive\Escritorio\database>sqlite3 simulador.db
SQLite version 3.30.0 2019-10-04 15:03:17
```

Fuente: Elaboración propia, 2020.

Al crear la base de datos como fue mencionado anteriormente se generará el archivo en el path donde se ejecutó el comando teniendo el siguiente resultado.

Figura 94: Archivo de la base de datos.

Name	Date modified	Type	Size
 simulador.db	8/6/2020 3:34 PM	Data Base File	20 KB

Fuente: Elaboración propia, 2020.

Habiendo finalizado la creación de la base de datos en nuestro sistema con el cual ya podremos añadir tablas y sus respectivas relaciones para ser utilizadas en el simulador de tiro con fusil de asalto. Con este punto realizado procederemos a implementar el siguiente punto de este objetivo.

3.5.3. Estableciendo las tablas y relaciones de la base de datos.

Para la realización de este punto, utilizaremos varios comandos de SQLite.

Tendremos que crear dos tablas, una de ellas será la de usuarios y la otra será la de puntuación en la cual se tendrán varios campos que se mostraran en la tabla estadística del simulador de tiro con fusiles de asalto. Para crear las tablas se realizará lo siguiente.

Figura 95: Creación de tabla “Usuarios”

```
sqlite> Create table usuarios(  
...> userId integer primary key,  
...> nombre text not null unique,  
...> fechaCreacion text default current_timestamp  
...> );
```

Fuente: Elaboración propia, 2020.

Como se puede observar en la Figura 95. El usuario tiene su identificador que es de tipo integer(entero) que será la clave primaria de la tabla, el nombre será de tipo text debido a que en sqlite no hay strings y solo text, se le agregará el unique para que solo pueda el nombre sea unico y no exista duplicados. Para la fecha de creación se manejará un texto que captura el tiempo en el que se cree el registro.

Figura 96: Creación de tabla “Puntuación”

```
CREATE TABLE puntuacion(  
puntuacionId integer primary key,  
kills integer,  
precision real,  
tiempo text,  
userId integer,  
foreign key (userId)  
References usuarios (userId)  
ON DELETE CASCADE  
ON UPDATE NO ACTION  
);
```

Fuente: Elaboración propia, 2020.

En la Figura 96 podemos observar que la tabla puntuación tendrá su identificador el cual es puntuacionId que será de tipo entero y también será la clave primaria de la tabla. Los kills (aciertos) serán de tipo entero y estos registrarán la cantidad de aciertos en una sesión dentro del simulador de tiro con fusiles de asalto. Para la precisión se la declarara de tipo real que en sqlite es equivalente a un valor flotante,

este albergara la cantidad de aciertos con relación a disparos realizados dentro del simulador de tiro con fusiles de asalto. El tiempo será de tipo texto y este incluirá los minutos y segundos de duración de una sesión con el simulador de tiro con fusiles de asalto.

3.5.4. Definición de los parámetros estadísticos a ser tomados en cuenta para la tabla.

Para definir los parámetros estadísticos a ser tomados en cuenta para la tabla, nos basaremos en parte del diseño realizado anteriormente, la tabla deberá tener la cantidad el usuario que realizo el entrenamiento dentro del simulador de tiro con fusiles de asalto, los aciertos que el usuario realizo los cuales serían la cantidad de bajas que se obtuvieron en el simulador, la precisión que determinara la cantidad de aciertos con relación a los fallos y el tiempo que nos permitirá saber la duración dentro del simulador de tiro con fusil de asalto.

En resumen, tendremos los siguientes parámetros a tomar en cuenta:

- Usuario
- Aciertos
- Precisión
- Tiempo

Y con esto concluiríamos toda la parte lógica para comenzar a implementarlo dentro del motor de gráficos Unity, utilizando las herramientas provistas por el mismo.

3.5.5. Codificación de la tabla estadística en base al usuario y los parámetros dados anteriormente.

Para realizar la codificación de la tabla estadística utilizaremos los Game Objects de UI para la realización de esta dentro de Unity la cual tendrá en vista de una cámara con perspectiva del entorno generado dando una visión de mitad terreno y mitad cielo donde renderizaremos la tabla para que el usuario pueda ver sus resultados.

Los Game Objects de UI que se utilizaran para la implementación serán una cámara, Canvas, Table UI, Buttons y el scroll view. Siguiendo a esto tendremos la tabla estadística ya implementada. Ver Anexo H.

Teniendo como resultado la siguiente tabla, la cual es dinámica porque va obteniendo los valores de la BD mediante consultas y se van generando los campos como en la Figura 97. Antes de la ejecución del simulador la tabla tiene esta forma.

Figura 97: Tabla de puntuaciones antes de la ejecución del simulador



ID	Usuarios	Aciertos	Precision	Tiempo
----	----------	----------	-----------	--------

Fuente: Elaboración propia, 2020.

Entonces cuando se ejecuta el simulador este procede a realizar consultas en la BD y a crear los objetos de manera dinámica.

Figura 98: Tabla de puntuación en Unity



ID	Usuarios	Aciertos	Precision	Tiempo
3	Crisner	30	100	00:45
4	Crisner	30	100	00:45
1	Claros	45	98	01:18
2	Jose	50	98	01:18
5	Gutierrez	3	1	00:24

Fuente: Elaboración propia, 2020.

Como podemos observar en la Figura 98. Se implemento una tabla con los siguientes campos:

- ID

- Usuario
- Aciertos
- Precisión
- Tiempo

Se colocó 3 botones, uno para volver al menú principal, otro para reintentar la sesión de entrenamiento y la última para generar reportes.

3.5.6. Generación de los reportes por usuario.

Para la generación de reportes por usuario se mostrará una interfaz en la cual se hará un reporte general del usuario seleccionado.

Este UI de reporte estará realizada con los componentes de UI del Unity y enlazada al BD que creamos anteriormente, teniendo el siguiente resultado como se podrá observar en la Figura 98.

Podemos observar lo que contiene la UI de reportes que a su vez es la UI de puntuaciones, la cual tendrá un buscador en el cual se podrá elegir el usuario del que se requiera el reporte, al seleccionar el usuario, habrá una sección donde se pueda ver cada una de las sesiones y su fecha respectiva.

Se mostrarán los 3 indicadores de la tabla, Aciertos, precisión y tiempo, en cada una de ellas se mostrará en grande los valores obtenidos en la sesión seleccionada. La interfaz tendrá un gráfico de barras que tendrá las sesiones y 1 de los indicadores, ya sea aciertos, precisión o tiempo.

Figura 99: Reporte generado por usuario.

	A	B	C
1	Crisner		
2			
3			
4			
5	precision	aciertos	tiempos
6	0	1	0:24
7	0	1	0:24
8	0	1	0:24
9	0	1	0:24
10	0	1	0:24
11	0	1	0:24
12	0	1	0:24
13	0	1	0:24
14	0	1	0:24
15	0	1	0:24
16	0	1	0:24
17	0	1	0:24
18	0	1	0:24
19	0	1	0:24
20	0	1	0:24
21	0	1	0:24
22	0	1	0:24
23	0	1	0:24
24	0	1	0:24
25	0	1	0:24
26	0	4	0:24
27	1	2	0:24
28	1	4	0:24
29	0	3	0:24
30	0	4	0:24
31	0	3	0:24
32	0	1	0:24
33	0	5	0:24
34	0	4	0:24
35	0	4	0:24
36	0	6	0:24
37	0	3	0:24
38			
	Crisner-10-23-2020		

Fuente: Elaboración propia, 2020.

Para ver el código de la generación de reportes ver Anexo I.

3.6. MODELACIÓN 3D DE LOS FUSILES PROPUESTOS (FN FAL, GALIL AR Y AK-47).

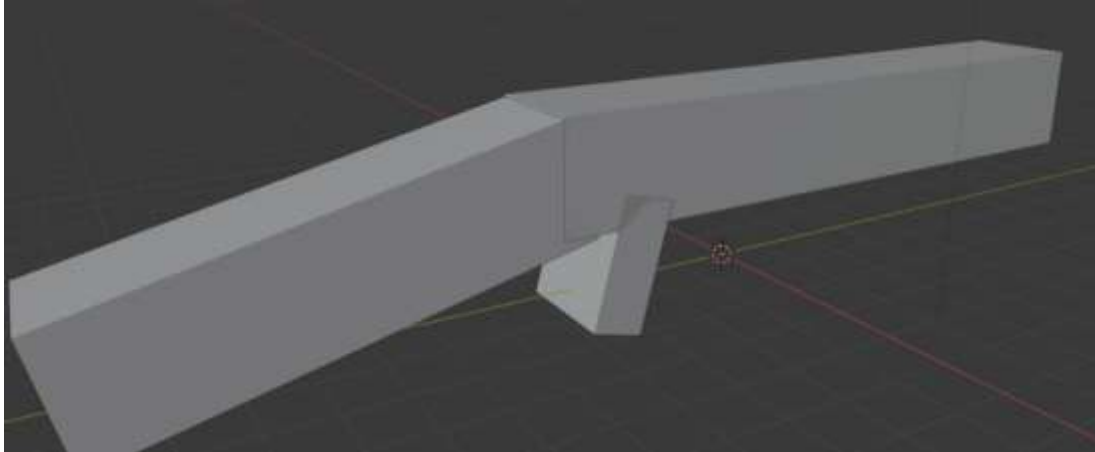
Para el modelado en 3d de los objetos que pertenecerán al simulador de tiro con fusil se utilizó la herramienta “Blender” la cual nos permite modelar objetos 3d y exportarlo para motores gráficos.

3.6.1. Diseño de los fusiles FN FAL, GALIL AR y AK-47.

En cuanto al diseño se terminó el modelado del fusil de asalto AK-47, Dividiéndolo por partes para poder utilizarlo en el simulador y generar animaciones interactivas.

Todo el fusil AK-47 fue modelado a partir de un objeto base siendo el mismo un prisma rectangular el cual fue cortado y pulido para ir obteniendo detalles como se mostrará a continuación.

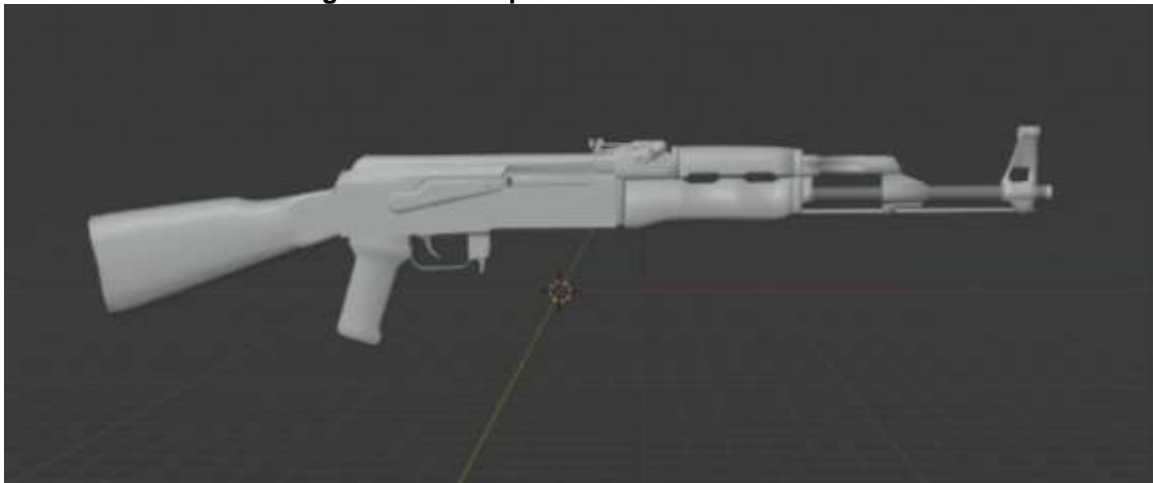
Figura 100: Cuerpo básico del fusil



Fuente: Elaboración propia, 2020.

Con la base inicial de fusil se procedió a ir añadiendo detalles utilizando la herramienta de Extrude y sculpting, con lo que se procedió a finalizar el modelado, A continuación, se mostrará el modelado del fusil AK-47.

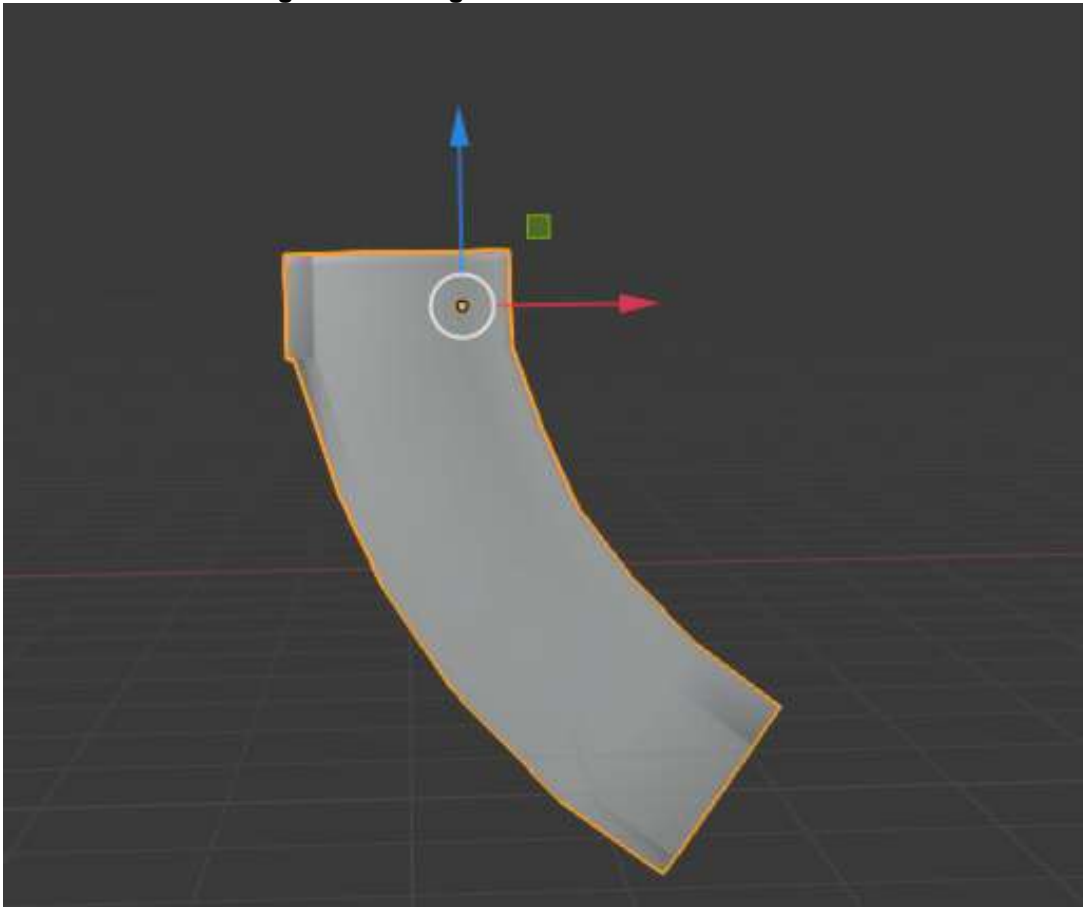
Figura 101: Cuerpo del fusil de asalto AK-47.



Fuente: Elaboración propia, 2020.

De igual manera se fue realizando el modelado y esculpido de las otras dos partes fundamentales para el arma, siendo el cargador y el cerrojo del fusil de asalto AK-47.

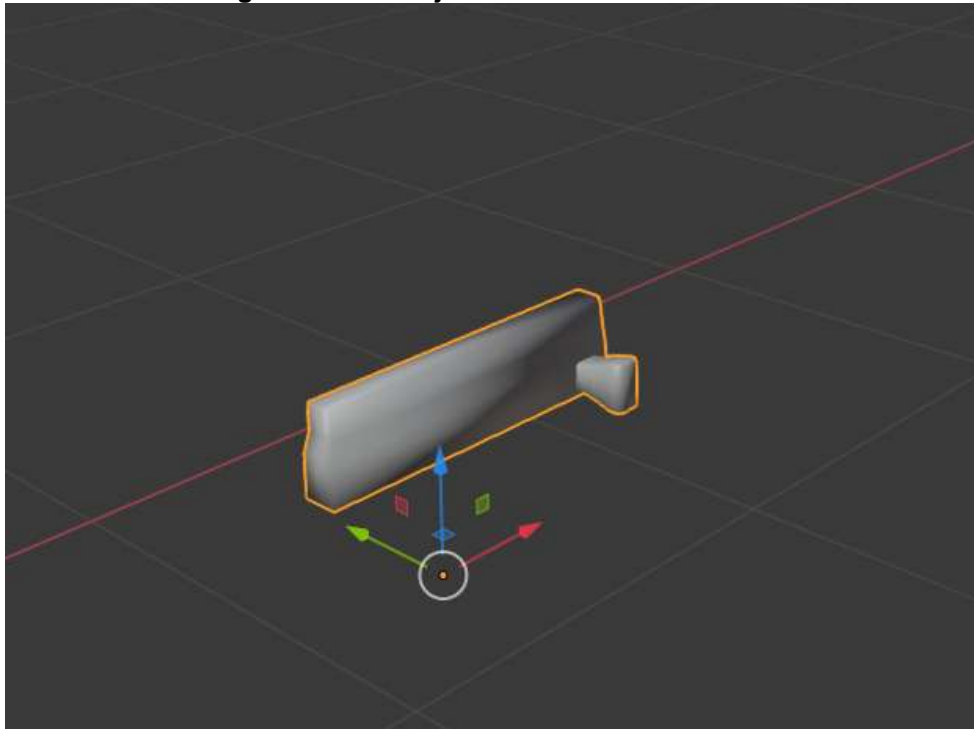
Figura 102: Cargador del fusil de asalto AK-47



Fuente: Elaboración propia, 2020.

Cada objeto modelado con Blender será exportado como un .obj el cual es una extensión compatible con el motor de gráficos de Unity, con los cuales nosotros podremos unirlos para crear un prefab, que es un prefabricado, en este caso al estar separados cada uno de los objetos podrán interactuar de manera independiente.

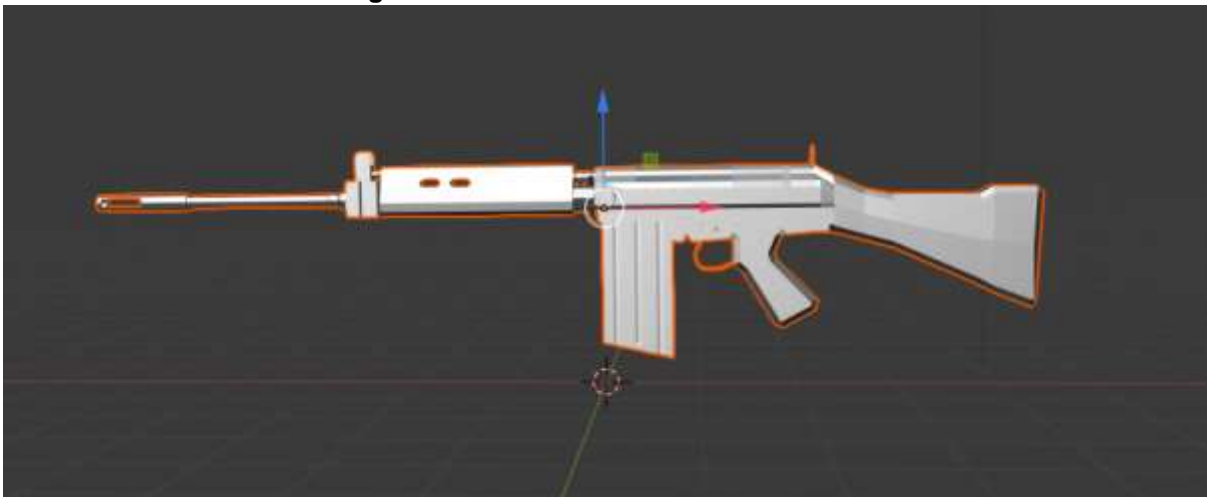
Figura 103: Cerrojo del fusil de asalto AK-47



Fuente: Elaboración propia, 2020.

El modelado del fusil FN FAL se realizó de igual forma que el fusil AK-47 y fue exportado en obj. Para su uso en el motor gráfico.

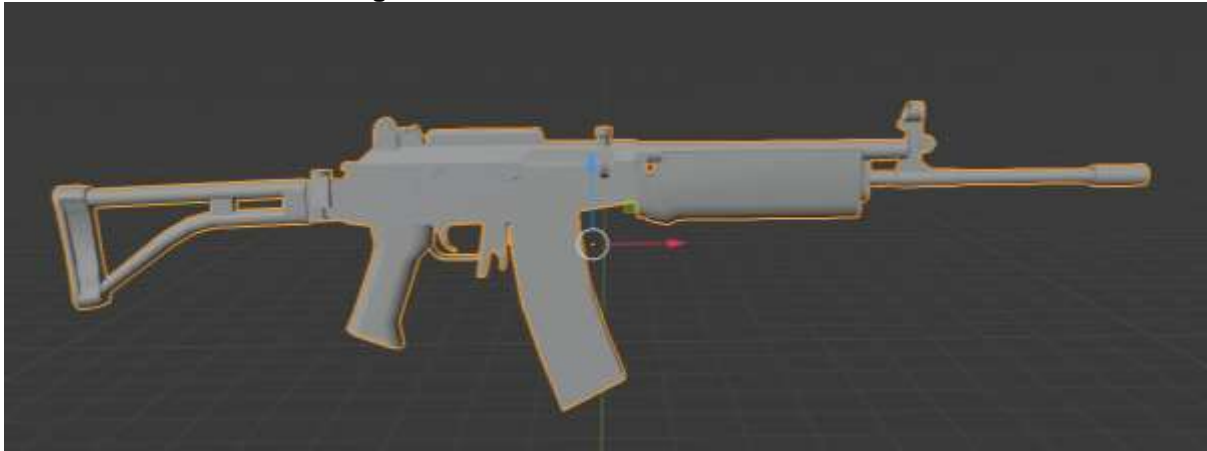
Figura 104: Modelado 3d del fusil FN FAL



Fuente: Elaboración propia, 2020.

Como se mencionó anteriormente este fusil fue modelado utilizando la menos cantidad de polígonos para obtener un objeto liviano.

Figura 105: Modelado del fusil GALIL AR



Fuente: Elaboración propia, 2020.

De igual forma este fusil fue modelado inicialmente con un conjunto de polígonos rectangulares creando la forma básica y empezando a realizar cortes para los detalles del fusil, tomando una imagen real del fusil como referencia. Con esto se tendría todos los modelos 3D de las armas para proceder a realizar el texturizado de las mismas.

3.6.2. Aplicación de las texturas respectivas en los modelos 3D de los fusiles.

Para el aplicado de texturas Blender tiene una herramienta para la gestión de texturas en el cual se pueden editar y crear texturas para todo tipo de objetos. A continuación, se mostrará las texturas de cada uno de los fusiles.

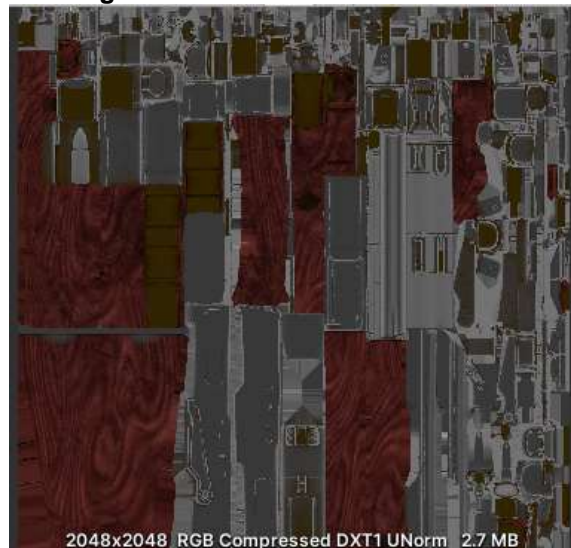
Figura 106: Aplicado de textura en el fusil AK-47



Fuente: Elaboración propia, 2020.

Como se puede observar, ya se aplicó la textura respectiva al objeto 3d, a continuación, se mostrará la textura que se aplicó al fusil de asalto AK-47.

Figura 107: Textura del fusil AK-47



Fuente: Elaboración propia, 2020.

Esta textura tiene los detalles del arma, las texturas son como una mascaró que cuando se aplica un objeto, el motor de gráficos lo aplica como una manta sobre el objeto, mientras más pintado, polígonos y detalles tengan el modelo 3d la mascaró o textura del objeto será más compleja.

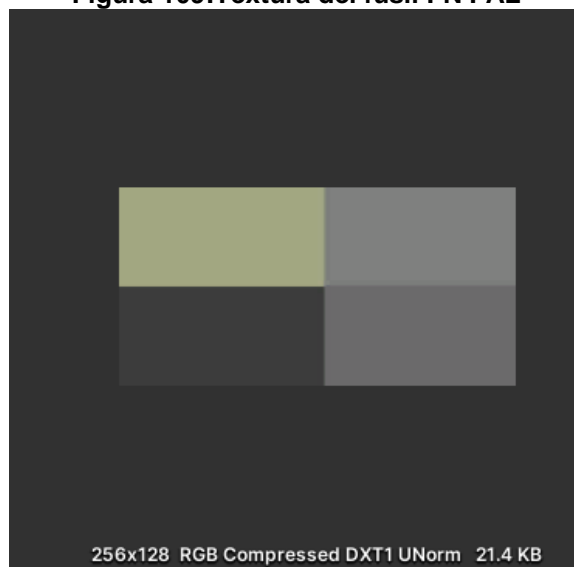
Figura 108: Textura aplicada al fusil FN FAL



Fuente: Elaboración propia, 2020.

De igual forma en el caso de fusil FN FAL la textura será más sencilla porque solo se aplicó el texturizado en las partes de la culata y parte frontal del fusil en el resto solo vario un poco el tono gris por defecto del objeto. La textura de este fusil es la siguiente.

Figura 109:Textura del fusil FN FAL



Fuente: Elaboración propia, 2020.

Como podemos observar en este ejemplo la textura es mucho más sencilla debido a la menor cantidad de polígonos que tiene el objeto.

Figura 110: Textura aplicada al fusil GALIL AR



Fuente: Elaboración propia, 2020.

El fusil fue modelado de la misma manera que el resto, inicialmente con objetos básicos representando la forma inicial, y detallando a medida que se hacen los cortes para llegar a representar el fusil, a su forma más semejante a la imagen de referencia.

Figura 111: Textura del fusil GALIL AR



Fuente: Elaboración propia, 2020.

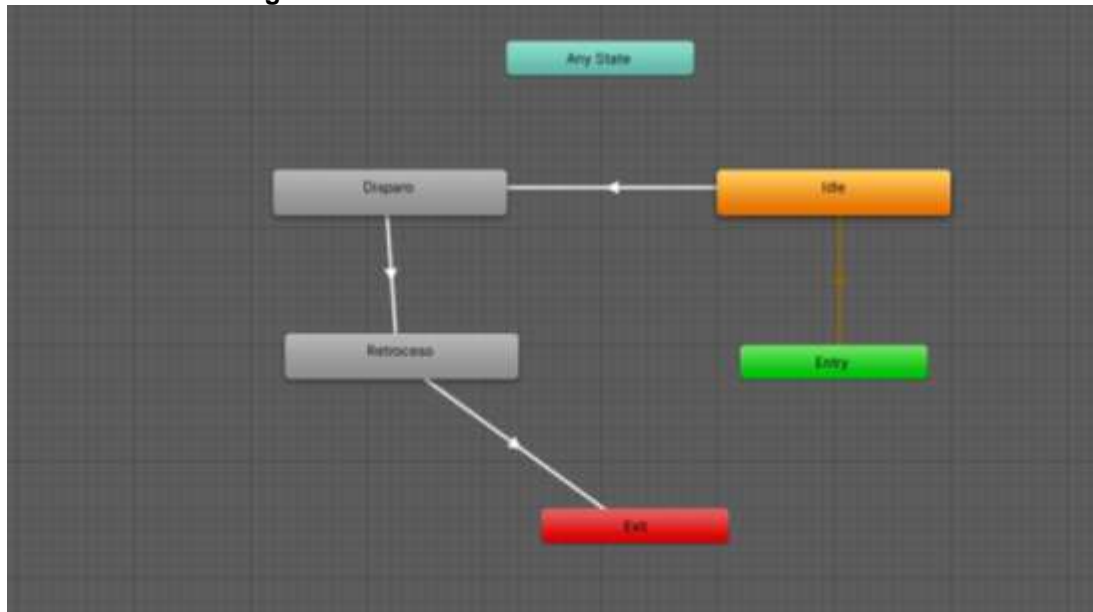
De igual forma que el AK-47 el pintado del arma y la cantidad de detalle dará la complejidad de la textura, esta textura es generada por Blender y será la máscara del objeto 3D en el motor de gráficos Unity. Con eso se tendrían las texturas de cada uno de los fusiles respectivos.

3.6.3. Codificación del funcionamiento de las animaciones de los modelos 3D.

Para la codificación de las animaciones de los modelos 3D se utilizará el animator que es una herramienta que incluye un lenguaje gráfico para la creación de animaciones de objetos, acá normalmente diagrama un flujo. Las animaciones debido a que se está utilizando la realidad virtual serán menos de lo normal, porque al ser VR las acciones las realizas con el headset.

Para la animación del fusil tendremos la siguiente lógica para realizarlo en el simulador de tiro con fusiles de asalto.

Figura 112: Animación de retroceso de las armas.



Fuente: Elaboración propia, 2020.

Se tendrá una entrada, luego a esto pasaremos a un estado idle, un estado idle es un estado en reposo, normalmente en un estado idle el objeto tiende a variar su movimiento un poco, siguiente a esto veremos que si ocurre un disparo sucederá la animación de retroceso al finalizar esta terminará y tendrá que haber otra entrada para obtener de nuevo la animación.

3.7. VALIDACIÓN DEL SISTEMA MEDIANTE UN PLAN DE QUALITY ASSURANCE (QA)

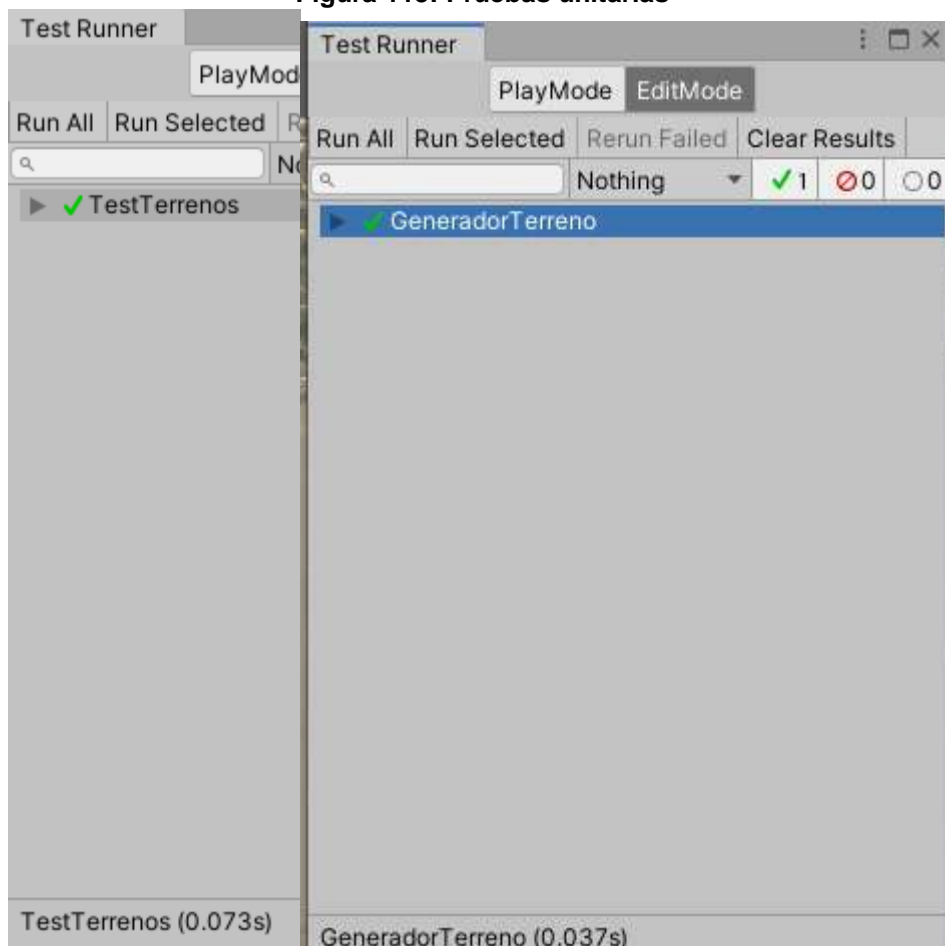
Para la validación del sistema mediante un plan de QA se utilizarán algunas herramientas que nos provee Unity para la realización de pruebas. Las cuales las iremos desglosando en los siguientes puntos.

3.7.1. Realización de las pruebas de testeo unitario al simulador.

Para el testeo unitario Unity tiene la herramienta del test runner en el cual puedes hacer una ejecución del script que quieras testear y aumentar tus propios test en cada script. Para esto utilizamos nuestros scripts más importantes que son los de la generación del terreno y generación de vegetación aleatoria.

Teniendo los siguientes resultados:

Figura 113: Pruebas unitarias



Fuente: Elaboración propia, 2020.

Como podemos observar, cada uno de los scripts fue ejecutado correctamente y con tiempos de compilación bajos, Test Terrenos fue compilado exitosamente en 0.073s y generador Terreno fue compilado exitosamente en 0.037s teniendo éxito en los módulos probados.

Tabla 12: Tiempos de ejecución

Test	Tiempo (s)
Terrenos	0.073
Generación Terreno	0.037
BD	0.051
SQLiteController	0.065
PlayerController	0.012
SecondHand	0.049
Grab	0.034
SoldierGeneratoController	0.051
ApplyDamage	0.048
LaserInput	0.041
Gun	0.052
GUIcontroller	0.038
VRUinput	0.036
Promedio	0.045

Fuente: Elaboración propia, 2020.

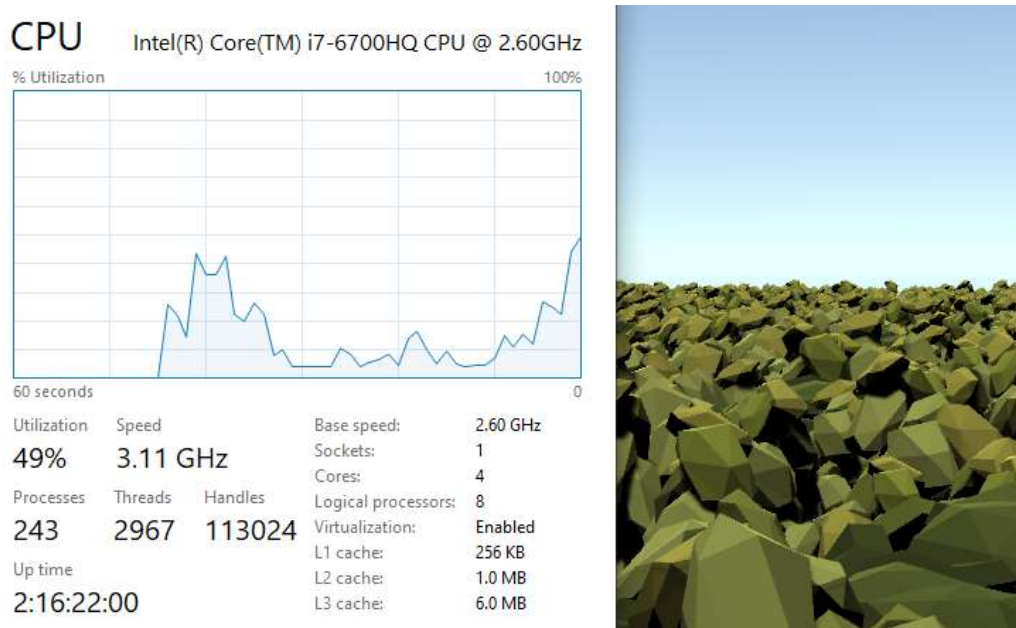
3.7.2. Realización de las pruebas de estrés al simulador.

Las pruebas de estrés realizadas se basan en las pruebas del sistema teniendo en cuenta que se evalúan las entradas y salidas como se suele hacer en pruebas de caja negra, con esto se podrá evaluar el rendimiento del software desarrollado dentro de un entorno (S.O) y cómo se comporta este en ciertos tipos de hardware.

Para las pruebas de estrés al simulador, se utilizaron de igual manera las herramientas provistas por Unity en las cuales se tuvo el siguiente resultado en una maquina con el siguiente hardware:

Procesador. – El procesador donde fue testeada la maquina es un Intel i7 de 6ta generación 6700 hq corriendo a una frecuencia estable de 3.1 GHz

Figura 114: Frecuencia del procesador máquina de prueba.



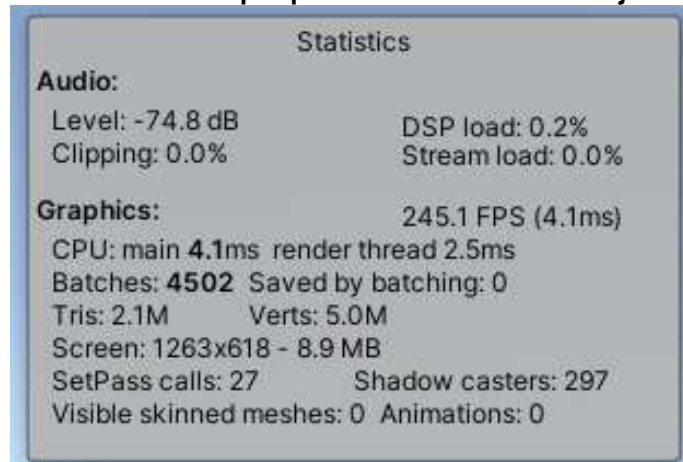
Fuente: Elaboración propia, 2020.

RAM. – La máquina donde se realizó la prueba de estrés tiene 16 GB RAM.

GPU. - La máquina donde se realizó la prueba de estrés tiene una GTX 1060 de 6GB de video dedicado y soporte para las tecnologías VR garantizado por NVIDIA.

Teniendo un panorama de la maquina donde se ejecutó la prueba, veremos el resultado.

Figura 115: Estadísticas obtenidas por parte del hardware en la ejecución del simulador.



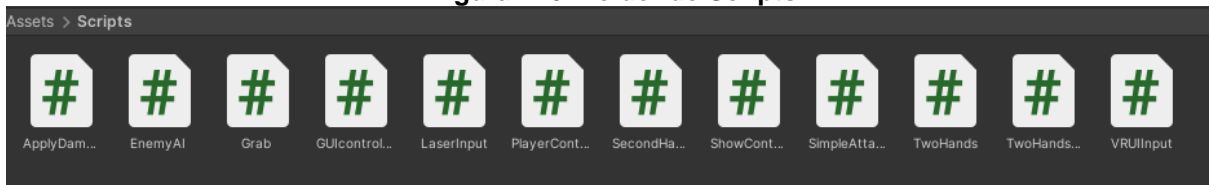
Fuente: Elaboración propia, 2020.

Como podemos observar en la Figura 115. En cuanto a los gráficos se obtienen un promedio de 241 FPS como media lo que significa que estamos obteniendo una tasa de frames constantes arriba de los 60 lo cual es muy bueno, pero al estar utilizando el oculus rift tendremos que mantener una media de frames arriba de los 144 FPS porque la pantalla integrada al oculus rift es de 144 Hz entonces para poder sincronizar la tasa de refresco de la pantalla con la del simulador se tendrá que garantizar más de 144 FPS para que el usuario pueda tener una buena experiencia con el simulador de tiro con fusiles de asalto.

3.7.3. Realización del test de integración a los módulos probados en el testeo unitario.

Para la realización de este punto se tomará en cuenta la metodología de caja gris, esta prueba nos permitirá validar el funcionamiento entre dos o más módulos implementados, dentro del margen de esta prueba tendremos un módulo externo el cual será la BD y se probará su integración con el simulador de tiro con fusiles de asalto. A continuación, se mostrarán los Scripts relevantes.

Figura 116: Folder de Scripts



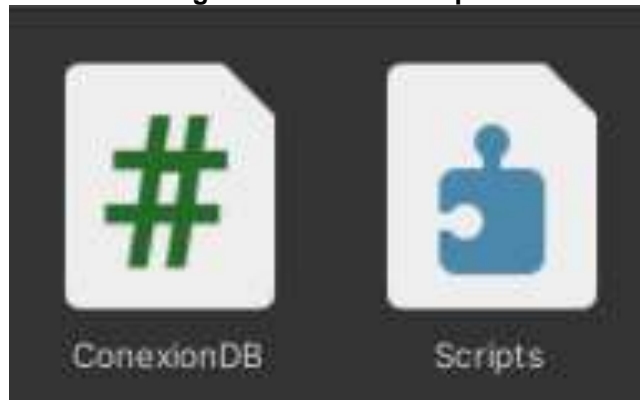
Fuente: Elaboración propia, 2020.

De igual forma que en las pruebas unitarias, testaremos si este módulo funciona correctamente debido a que es el encargado de integrar la DB (DataBase) con Unity.

Antes de ejecutar cualquier prueba se deberá dar una referencia de assembly en la carpeta de scripts, de esta manera el IDE podrá encontrar desde la dirección de los test los scripts actuales que se tienen.

La siguiente imagen nos muestra el Script ConexionBD que se encarga de la conexión de la BD y el respectivo assembly de la carpeta Scripts.

Figura 117: Folder scripts



Fuente: Elaboración propia, 2020.

La siguiente imagen nos muestra la carpeta Tests en la cual nosotros tendremos dos subcarpetas, la de EditMode y PlayMode las cuales contendrán los tests de los diferentes scripts, como se podrá observar la carpeta de EditMode cuenta con su respectivo assembly para que este pueda referenciarse con la carpeta de Scripts y se puedan ejecutar las pruebas.

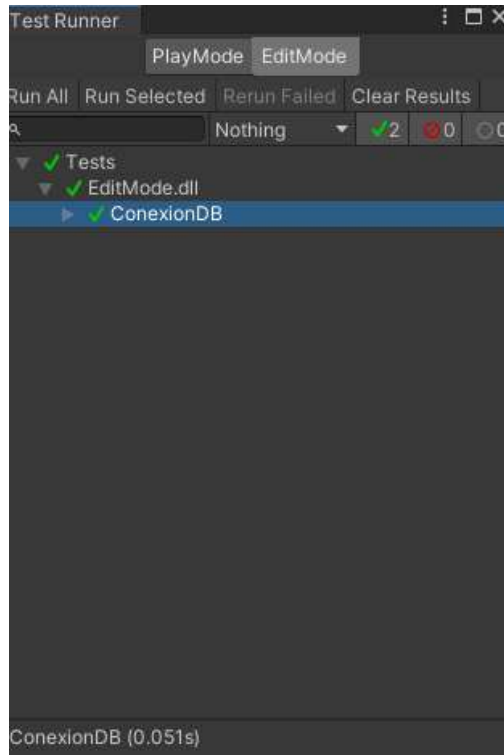
Figura 118: Folder tests



Fuente: Elaboración propia, 2020.

Como podemos ver en la siguiente imagen, nuestro módulo de integración funciona correctamente y por ende la integración del mismo es correcta.

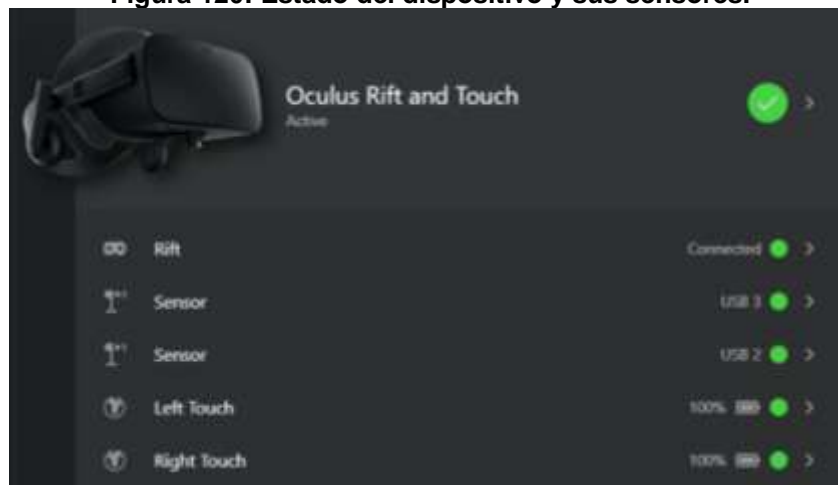
Figura 119: Prueba al módulo de conexión de la BD



Fuente: Elaboración propia, 2020.

También para verificar el funcionamiento del hardware se tiene el software propio de oculus:

Figura 120: Estado del dispositivo y sus sensores.



Fuente: Elaboración propia, 2020.

La conexión con el oculus rift utiliza un total de 4 puertos físicos, 2 puertos USB para los sensores infrarrojos, 1 puerto USB para la conexión del headset con la computadora y un puerto HDMI para el envío y recepción de imágenes y sonido.

Para más detalle del hardware externo ver Anexo “K” y Anexo “L”.

3.8. DEMOSTRACIÓN DE HIPÓTESIS.

a. Hipótesis

El Desarrollo de un simulador para el entrenamiento de tiro con fusiles de asalto FN FAL, GALIL AR y AK-47, bajo entornos dinámicos y blancos móviles usando realidad virtual permitirá realizar mayor cantidad de prácticas de tiro con diferentes fusiles reduciendo los costos de los entrenamientos de tiro con fusiles de asalto del BATALLÓN DE POLICÍA MILITAR III ESTEBAN ARCE”.

b. Hipótesis Nula

Nula (H_0) = El uso del simulador para el entrenamiento de tiro con fusiles de asalto no disminuirán los costos de operación.

c. Hipótesis Alternativa

Alternativa (H_1) = El uso del simulador para el entrenamiento de tiro con fusiles de asalto disminuirá el costo de operación.

d. Chi cuadrado

Para mayor detalle de los costos que se tomaran en cuenta en el cálculo del chi cuadrado ver el Anexo E.

Usando frecuencias en Fr.

Tabla 13: Estadísticas tabuladas: simulador, costo operación

Simulador	Munición 5.56mm	Munición 7.62mm	Todo
Con Simulador	324	324	648
	319.6	328.4	
	0.24812	-0.24474	
Sin Simulador	40095	41217	81312
	40099.4	41212.6	
	-0.02215	0.02185	
Todo	40419	41541	81960

Fuente: Elaboración propia, 2020.

Contenido de la celda

Conteo

Conteo esperado

Residuos estandarizados

Valor $P > 0.05$

$0.726 > 0.05$

Se rechaza la hipótesis nula y se acepta la hipótesis alternativa, Se demuestra que el simulador de tiro con fusiles de asalto disminuirá el costo de operación

CAPÍTULO IV: ANÁLISIS DE VIABILIDAD



CAPITULO 4: ANÁLISIS DE VIABILIDAD

4.1. VIABILIDAD TÉCNICA.

Para el análisis de viabilidad técnica inicialmente se realizarán el desarrollo de tablas en las cuales se especificarán los requerimientos mínimos necesario para la realización del sistema y los requerimientos mínimos necesarios para la implementación del mismo.

Para el desarrollo del sistema se requerirá mínimamente lo siguiente.

Tabla 14: Requerimientos mínimos para el desarrollo del sistema.

NOMBRE	REQUERIMIENTO MÍNIMO	TIPO DE RECURSO
Sistema Operativo	Windows 10	Software
Herramienta de programación	Visual Studio 2019 Community	Software
Herramienta de modelado en 3d	Blender v2.83	Software
Motor grafico	Unity 2020.1.2f1	Software
Procesador	Intel i3-6100/AMD Ryzen 3 1200, FX4350 o superior	Hardware
Memoria RAM	8 GB de RAM o más	Hardware
Tarjeta de video	NVIDIA GTX 1060/AMD Radeon RX 480 o superior	Hardware
Disco Duro	512 GB SSD	Hardware
Sistema de realidad virtual	Oculus Rift	Hardware
Puertos USB para Rift	Un puerto USB 3.0 y dos puertos USB 2.0	Hardware
Salida de vídeo (Rift)	Salida de vídeo HDMI 1.3 compatible	Hardware

Fuente: Oculus Rift, 2020.

El sistema operativo elegido en este caso es Windows 10 debido a que todo el SDK (Kit de Desarrollo de Software) del oculus rift, otro de los factores es los problemas de compatibilidad de los drivers de la tarjeta gráfica en Linux, debido a esto es necesario utilizar Windows para acceder a toda la capacidad de la tarjeta gráfica ya que estas están optimizadas para funcionar en Windows.

En cuanto a la herramienta de programación se hará uso de Visual Studio 2019 Community debido a que es de uso gratuito y es el IDE recomendado por Unity.

Para la herramienta de modelado en 3d se utilizará el programa open source denominado Blender, el cual hoy en día es muy popular porque ofrece herramientas a nivel profesional y de manera gratuita. Ahora, todo el hardware mencionado debería permitir el desarrollo del simulador sin ningún problema, ya que estas partes de hardware rinden lo suficiente para esta tarea.

Teniendo en cuenta todo esto, al finalizar el desarrollo, se requerirá lo siguiente para la implementación del sistema en la institución.

Tabla 15: Requisitos recomendados para el uso del sistema.

NOMBRE	REQUISITOS RECOMENDADO	TIPO DE RECURSO
Sistema Operativo.	Windows 10	Software
Software de gestión de realidad virtual.	Oculus	Software
Procesador.	Intel i5-4590/AMD Ryzen 5 1500X o superior	Hardware
Memoria RAM.	8 GB de RAM o más	Hardware
Tarjeta de video.	NVIDIA GTX 1060/AMD Radeon RX 480 o superior	Hardware
Disco Duro.	512 GB M.2	Hardware
PSU	650W Corsair 80+ Gold	Hardware
Sistema de realidad virtual.	Oculus Rift	Hardware
Puertos USB para Rift	Un puerto USB 3.0 y dos puertos USB 2.0	Hardware
Salida de vídeo (Rift)	Salida de vídeo HDMI 1.3 compatible	Hardware
Teclado.	Teclado Genérico	Hardware
Mouse.	Mouse Genérico	Hardware
Monitor.	Monitor 60hz panel IPS	Hardware

Fuente: Oculus Rift, 2020.

El ejecutable que se obtendrá al finalizar el desarrollo estará destinado para Windows 10, el Software de gestión de realidad virtual denominado Oculus es por el cual se puede realizar todas las configuraciones del oculus rift (Hardware), este software es obligatorio para el uso del oculus rift.

En cuanto al hardware se tendrán periféricos de entrada y de salida siendo los típicos mouses, teclado, monitor y el Oculus rift.

Como se puede observar en la tabla 13 existe un cambio de hardware en cuanto a la tarjeta gráfica en relación a la tabla 14. Esto es porque se tendrá que alcanzar una tasa mayor de frames para poder utilizar el simulador de manera óptima y necesariamente necesitaremos mínimamente esa tarjeta gráfica.

4.2. VIABILIDAD ECONÓMICA.

En esta sección se realizará un análisis de los costos del sistema, tanto para hardware, software y el esfuerzo de realización que este simulador conlleva tomando en cuenta el esfuerzo intelectual y la programación realizada.

4.2.1. Estimación de costos de la empresa.

Para los costos tomaremos en cuenta el esfuerzo, software y hardware. Tomando en cuenta las tablas anteriores iniciaremos con sus costos respectivos.

Tabla 16: Costo del equipo para el uso del sistema.

NOMBRE	COSTO(Bs)
Windows 10	210.00
Intel i5 6700k	1100.00
16GB DDR4 3000 MHz	1100.00
NVidia RTX 2070 super 8GB	5460.00
512 GB M.2	770.00
650W Corsair 80+ Gold	600.00
Case	400.00
Oculus Rift	5110.00
Teclado Genérico	50.00
Mouse Genérico	30.00
Monitor 60hz panel IPS	700.00
COSTO TOTAL	15530.00

Fuente: Elaboración propia, 2020.

El costo del equipo a implementar será de 15530 Bs para poder hacer uso del simulador.

Para el estimado del costo en base al esfuerzo utilizaremos el método Análisis de puntos de función.

4.2.2. Estimación de esfuerzo

Para este cálculo tomaremos un como dato del proyecto unas 1400 líneas de código aprox. Y con esto podremos obtener la estimación del esfuerzo.

$$ESF = 3 * ML^{1.12}$$

$$ESF = 3 * (1.4)^{1.12} = 3.4741 \approx 4 \text{ Personas/Mes}$$

Donde:

$$ESF = \text{Esfuerzo} \left(\frac{\text{Personas}}{\text{Mes}} \right)$$

ML = Miles de lineas de código

Entonces, tenemos como estimación del esfuerzo que, para la realización del simulador de tiro con fusiles de asalto, se necesitaran 4 personas/mes para el desarrollo del mismo.

4.2.3. Estimación del costo

Para la estimación del costo se realizarán los siguientes cálculos:

$$C = ESF * CHM$$

$$C = 3.4741 * 4900$$

$$C = 17024Bs$$

Donde:

C = Costo del proyecto

CHM = Costo Hombre Maquina (Salario)

4.2.4. Beneficio costo.

Para tener una idea de cuál será el beneficio en relación al costo hay que analizar algunos que detalles que ya fueron mencionados anteriormente en este documento, como ser el costo de la munición utilizada para las practicas tiro, y la cantidad utilizada, que hacen que los costos sin el sistema asciendan a un monto de 40000 Bs aproximadamente por cada entrenamiento de tiro realizado.

Tabla 17: Costo del entrenamiento de un batallón con munición 7.62mm

Descripción	Cantidad	Costo
-------------	----------	-------

Munición 7.62mm	8019	41217.66 Bs.
-----------------	------	--------------

Fuente: Elaboración propia, 2020.

Como se puede observar el costo de un entrenamiento para un batallón con munición de 7.62mm tiene un costo aproximado de 41217.66 Bs.

Tabla 18: Costo del entrenamiento de un batallón con munición 5.56mm

Descripción	Cantidad	Costo
Munición 5.56mm	8019	40095 Bs.

Fuente: Elaboración propia, 2020.

De igual forma para un entrenamiento de un batallón con munición de 5.56mm tiene un costo de 40095Bs.

En la siguiente tabla se mostrarán los costos del simulador y como este podría beneficiar en caso de su implementación.

Tabla 19: Costo con el simulador de tiro con fusiles.

Descripción	Costo
Costo Desarrollo	17024 Bs.
Costo Equipo	15530 Bs.
Costo Total	32554 Bs.

Fuente: Elaboración propia, 2020.

Como se pudo observar haciendo la comparación de costos, con el simulador implementado se podrán realizar entrenamientos ya sea con munición 7.62mm o munición 5.56mm y los entrenamientos que se podrán realizar no tendrán un límite más que el tiempo invertido en el simulador.

Utilizando la fórmula de beneficio/costo tenemos que:

$$\frac{\textit{Beneficio}}{\textit{Costo}} = \frac{81312.66}{32554}$$

$$\frac{\textit{Beneficio}}{\textit{Costo}} = 2.4977$$

El resultado será mayor a uno, lo que significa que el simulador de tiro con fusiles de asalto AK-47, GALIL AR y FN FAL. Será rentable a corto y largo plazo.

CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES



CAPITULO 5: CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

- Se realizó un análisis del proceso actual de entrenamiento con fusil y su entorno de desempeño tomando en cuenta toda la información reunida, utilizando el método de entrevista al Cnl. DAEN Tomas Crisner Prado y el método de análisis bibliográfico sobre artículos y revistas de balística, en base a esto se generó el modelo general para la realización del simulador de tiro con fusiles de asalto.
- Se aplicó el método FDD (Feature Driven Development) a través del manejo de características de forma iterativa para visualizar de forma completa todo el sistema en donde se planifico el tiempo de desarrollo necesario para el simulador el cual fue cumplido satisfactoriamente.
- Para la implementación de los entornos dinámicos se obtuvieron las curvaturas del entorno mediante el GDEM accediendo a los repositorios geográficos de la NASA con los cuales se obtuvieron las imágenes satelitales de Cochabamba, realizando el parseo de la imagen y transformándolas a escala de grises para posteriormente ser transformadas a mapas digitales utilizando el algoritmo de K-Neighbors que permite realizar la renderización de manera automática.
- Para establecer las características del disparo en los climas diversos de acuerdo a los terrenos dinámicos, se recopiló información de los entornos para codificar los atributos del clima y la física de la trayectoria de disparo mediante Visual Studio utilizando el lenguaje C#.
- Se implementó una tabla estadística de tiro para observar la evolución del usuario diseñando una base de datos, siguiendo esto se creó la base de datos utilizando SQLite, tomando en cuenta los parámetros definidos en el documento, Aplicando una integración entre el motor gráfico y el archivo de la base de datos para obtener la generación de reportes por usuario que genera un historial de cada usuario y muestra sus puntuaciones.

- Se modeló en 3d los fusiles de asalto AK-47, FN FAL y GALIL AR utilizando la herramienta Blender con la cual se modeló y se realizó el texturizado correspondiente a cada uno de los objetos, para obtener un .fbx y .obj de los 3 fusiles mencionados que conectan al hardware de realidad virtual de manera íntegra.
- Se validó el sistema al finalizar su desarrollo mediante una serie de pruebas que son las siguientes:
 - Pruebas unitarias: Se verificó el funcionamiento de cada módulo del sistema con un tiempo promedio de 0.045s por cada ejecución.
 - Pruebas de estrés: Se verificó el rendimiento del software en un equipo de hardware aumentando la cantidad de 800000 objetos renderizados en el cual el resultado fue a una baja de los cuadros por segundo.
 - Prueba de integración: Se verificó la integración de los módulos respectivos de software con el hardware de realidad virtual.

Finalmente se llegó a la conclusión de que con el desarrollo de un simulador para el entrenamiento de tiro con fusiles de asalto FN FAL, GALIL AR Y AK-47, bajo entornos dinámicos y blancos móviles usando realidad virtual se logró comprobar la reducción de los costos y aumentar la cantidad de entrenamientos realizados, esto en base a la disponibilidad de tiempo del usuario.

5.2. RECOMENDACIONES

- No recomiendo la utilización del método FDD en un desarrollo individual debido a que el método toma en cuenta a varios equipos, delegando actividades a cada uno de ellos, De esta manera al desarrollar de manera individual se llegara a una sobrecarga de trabajo.
- Se recomienda continuar el trabajo con una modalidad de tesis en conjunto para tener un mayor alcance en cuanto a funcionalidades del sistema y evitar la sobrecarga de trabajo.
- Se recomienda realizar un módulo para la sectorización de las imágenes obtenidas mediante el GDEM, para la personalización de terrenos de manera automática.

- Se recomienda añadir más climas dado que esta tesis de grado está delimitada en la zona de los valles, añadiendo sus respectivos atributos de cada clima.
- Se recomienda aumentar reportes grupales y gráficas de evolución grupales, para que de esta manera se pueda analizar de manera grupal la eficiencia de los usuarios.
- Se recomienda añadir un overlay de mapa para mostrar la ubicación dentro del terreno al combatiente.
- Se recomienda trabajar sobre hardware de realidad virtual disponible a la fecha en la que retome este trabajo de grado.
- Se recomienda automatizar el cargado de modelos 3d de fusiles como parte de una mejora a este sistema, para futuros proyectos.
- Se recomienda generar este mismo proyecto implementando múltiples usuarios en una intranet para que de esta manera se pueda entrenar por escuadras simulando operaciones de combate grupal.
- Se recomienda realizar un nuevo diseño de interfaces aplicando conocimientos de diseño de UX y UI.
- Se recomienda la implementación de hardware externo para la simulación física del arma en sintonía con el software, para ofrecer una experiencia mucha más cercana a la realidad.
- Se recomienda mejorar la dinámica del VR añadiendo más animaciones a los combatientes, y para el manejo de las armas.
- Se recomienda añadir sonido ambiental para una mejor inmersión dentro del simulador de tiro con fusiles de asalto.
- Se recomienda añadir localidades aleatorias con civiles dentro de lo que concierne a la generación del terreno.
- Se recomienda ampliar los terrenos a todas las regiones propias del país debido a la diversidad de terrenos existentes en Bolivia.
- Se recomienda añadir climas dinámicos durante la ejecución del simulador de tiro con fusiles de asalto.

- Se recomienda continuar con este trabajo de grado, debido a que aún se encuentra en una fase temprana de desarrollo, siendo que puede ser ampliada para tener mayor utilidad y provecho de este trabajo de grado.

BIBLIOGRAFÍA



BIBLIOGRAFÍA

REFERENCIAS BIBLIOGRAFICAS

- Bragado, I. M. (2003). *Física*. Valladolid, España: Universidad de Valladolid.
- IEEE. (2010). *IEEE Standar for Modeling and Simulation (M&S) High Level Architecture (HLA) -- Framework and Rules*. New York, USA.
- IEEE. (2014). *Guided to the Software Engineering Body of Knowledge* (Version 3.0 ed.). IEEE Computer Society.
- Kelton, W. D., Randall P. Sadowski, & Deborah A. Sadowski. (2001). *Simulation with Arena*. McGraw-Hill.
- Lampel, J. (2015). *THE BEGINNERS GUIDE TO BLENDER*.
- LaValle, S. M. (2019). *VIRTUAL REALITY*. (U. o. oulu, Ed.) Cambridge University Press.
- Opel, A. (2009). *Fundamentos de SQL* (Tercera edicion ed.). Mexico D.F, Mexico: McGRAW-HILL.
- Simons, B. (2013). *BLENDER MASTER CLASS*. San Francisco, California, USA: No Starch Press.
- Unity. (16 de 04 de 2019). *Unity Technologies*. Recuperado el 2 de Marzo de 2020, de Unity Documentation: <https://docs.unity3d.com/Manual/index.html>
- Richard Schmidt. (2013), *Software engineering architecture-driven software development*. Morgan Kaufmann, El Sevier. MA. USA
- Giancarlo Zaccone. (2015), *Python parallel programming cookbook*. Packt Publishing
- Sheetal Sharma. (2012). *Agile Processes and Methodologies: A Conceptual Study*. *International Journal on Computer Science and Engineering (IJCSE)*
- Jacob Olsen. (2004). Realtime Procedural Terrain Generation. Department of Mathematics And Computer Science (IMADA). University of Southern Denmark. Dinamarca.
- Sadhna Goyal (2007). *Agile Techniques for Project Management and Software Engineering*. Technical University Munich. Munich. Alemania.

REFERENCIAS WEBIBLIOGRAFICAS

- Pedro Sá Silva, António Trigo, João Varajão and Tiago Pinto, (2010), Simulation - Concepts and Applications. Recuperado el 27 de marzo de 2020 de: https://www.researchgate.net/publication/221002969_Simulation_-_Concepts_and_Applications
- Björn Möller, Mikael Karlsson, (2010), New Object Modeling Opportunities in HLA 4 Recuperado el 28 de marzo de 2020 de: https://www.researchgate.net/publication/331320274_New_Object_Modeling_Opportunities_in_HLA_4
- IEEE, Ahmed E. Hassan, (2015) A Survey on Load Testing of Large-Scale Software Systems, Recuperado el 29 de marzo de 2020 de: https://www.researchgate.net/publication/282551435_A_Survey_on_Load_Testing_of_Large-Scale_Software_Systems
- Ricki. G Ingalls, (2001), Introduction to simulation, Recuperado el 29 de marzo de 2020 de: https://www.researchgate.net/publication/221526292_Introduction_to_simulation
- Bhojaraju Gunjal, (2003), Database Management: Concepts and Design Recuperado el 29 de marzo de 2020 de: https://www.researchgate.net/publication/257298522_Database_Management_Concepts_and_Design, accedido
- Moses Okechukwu Onyesolu, (2011), Understanding Virtual Reality Technology: Advances and Applications, Recuperado el 30 de marzo de: https://www.researchgate.net/publication/221911335_Understanding_Virtual_Reality_Technology_Advances_and_Applications
- Ian Gorton, (2001), Understanding Software Architecture Recuperado el 29 de marzo de 2020 de: https://www.researchgate.net/publication/251164302_Understanding_Software_Architecture
- Bjarne Stroustrup, (1998), What is "Object-oriented Programming"? Recuperado el 28 de marzo de 2020 de: https://www.researchgate.net/publication/3246605_What_is_Object-oriented_Programming

ANEXOS



ANEXO “A” CARTA DE ACEPTACIÓN



S. STRIA. GRAL. No. 076/20

Objeto : Elevar Perfil

Anexo : De Referencia.

Cotapachi, Febrero 10 de 2020

Sr. Cnl. DAEN. Jason Sócrates Fuentes Flores
**DIRECTOR DE LA UNIDAD ACADEMICA COCHABAMBA DE LA ESCUELA
MILITAR DE INGENIERIA**

Presente. -

Señor Coronel:

Mediante la presente, reciba Ud. el saludo cordial y afectuoso a nombre del personal de Oficiales Superiores, Oficiales Subalternos, Suboficiales, Sargentos, Personal Administrativo, Soldados, Premilitares del RPM-3 "GRAL. ARZE" y mío en particular deseándole los mayores éxitos en las delicadas funciones que desempeña.

Con la finalidad de cumplir con requerimientos de modernización de nuestra Unidad Militar, se requiere la implementación de un simulador de tiro, para lo cual, se Autoriza al Sr. Carlos Andres Crisner Velarde, estudiante de la carrera de Ingeniería de Sistemas, para el **DESARROLLO DE UN SIMULADOR DE ENTRENAMIENTO DE TIRO CON FUSILES DE ASALTO FN, FAL, GALIL, AR Y AK-47, BAJO ENTORNOS DINAMICOS Y BLANCOS MOVILES USANDO REALIDAD VIRTUAL.**

Con este motivo, saludo al Señor Coronel con la seguridad de mi atenta y distinguida consideración.

**"EL MAR NOS PERTENECE POR DERECHO,
RECUPERARLO ES UN DEBER"**

RAR/ab.



Tcnl. DEM. Jorge Alfredo Meneses Serrano
COMANDANTE DEL RPM-3 "GRAL. ARZE"

ANEXO “B” CARTA DE CONFORMIDAD



S. STRIA. GRAL. No. 077/20

Objeto : Visto Bueno

Anexo : De Referencia.

Cotapachi, Noviembre 09 de 2020

Sr. Cnl. DAEN. Jason Sócrates Fuentes Flores
**DIRECTOR DE LA UNIDAD ACADEMICA COCHABAMBA DE LA ESCUELA
MILITAR DE INGENIERIA**

Presente.-

Señor Coronel:

Mediante la presente, reciba Ud. el saludo cordial y afectuoso a nombre del personal de Oficiales Superiores, Oficiales Subalternos, Suboficiales, Sargentos, Personal Administrativo, Soldados, Premilitares del RPM-3 "GRAL. ARZE" y mío en particular deseándole los mayores éxitos en las delicadas funciones que desempeña.

Una vez evaluado el trabajo realizado por parte del estudiante Carlos Andres Crisner Velarde, de la especialidad de Ingeniería de Sistemas, bajo el nombre de **"DESARROLLO DE UN SIMULADOR DE ENTRENAMIENTO DE TIRO CON FUSILES DE ASALTO FN, FAL, GALIL, AR Y AK-47, BAJO ENTORNOS DINAMICOS Y BLANCOS MOVILES USANDO REALIDAD VIRTUAL"**, y en vista de que los requisitos fueron cumplidos a cabalidad, se otorga el visto bueno por parte del RPM-3 "GRAL. ARZE".

Con este motivo, saludo al Señor Coronel con la seguridad de mi atenta y distinguida consideración.

**"EL MAR NOS PERTENECE POR DERECHO,
RECUPERARLO ES UN DEBER"**

BAR/ab.



Tcnl. DEM. Jorge Alfredo Meneses Serrano
COMANDANTE DEL RPM-3 "GRAL. ARZE"

ANEXO “C”
ENTREVISTA SOBRE EL
PROCESO DE
ENTRENAMIENTO DE
TIRO



ANEXO C “ENTREVISTA SOBRE EL PROCESO DE ENTRENAMIENTO DE TIRO”

Entrevista realizada al Cnl. DAEN. Tomas Crisner Prado para la recopilación de información sobre el proceso de entrenamiento de tiro realizado en el ejército.

Para la realización de la instrucción de tiro con fusil, se realizan 7 lecciones de tiro.

Primera lección:

Tiro de céreo

- En esta lección luego de la explicación teórica de las formas de puntería. El instructor y el tirador se dirigen al polígono de tiro en el cual se pone en práctica el “céreo del arma”; ósea las correcciones de acuerdo al ojo de puntería del tirador y al sistema de mira del arma.

La finalidad de esta lección de tiro es para que el tirador encuentre la correcta alineación de la imagen entre el blanco y el sistema de puntería de acuerdo a su capacidad física de brazos de apoyo, ojo (alza y guion del fusil). De esta manera el tirador determina su tipo ideal de puntería.

Para la segunda lección se realiza tiro de tendido con apoyo utilizando 5 cartuchos.

Para la tercera lección se realiza tiro tendido sin apoyo utilizando 5 cartuchos.

Para la Cuarta lección se realiza tiro de sentado /arrodillado con 5 cartuchos.

Para la Quinta Lección de pie con 5 cartuchos.

Para la sexta lección de tiro nocturno al bulto con 5 cartuchos y para la séptima lección tiro de acción y reacción con 5 cartuchos, total cartuchos por Soldado 33.

ANEXO "D"

COSTO DE UNA WORKSTATION



ANEXO D “COSTO DE UNA WORKSTATION”

Costo de una workstation con garantía de 8 años, con un consumo de electricidad de 650W.

Cotización de componentes			
Componentes	Producto	Cantidad	Costo(usd)
Procesador	Intel I7 9900k	1	530
Placa Madre	Asus ROG Z390 h	1	250
Tarjeta grafica	ASUS RTX 2080 super	1	990
memoria RAM	G.Skill 3000MHz 8GB	4	330
Almacenamiento P.	M.2 Crucial 512GB	1	110
Almacenamiento S.	1TB SSD	1	130
PSU	650W Corsair platinum	1	150
Refrigeración	Nzxt kraken	1	150
Case	Nzxt h501 elite	1	150
		Total	2790

Los siguientes componentes fueron cotizados en tiendas de componentes de cbba.

ANEXO “E”
CALCULO
COMPLEMENTARIO
HIPOTESIS



ANEXO E “CALCULO COMPLEMENTARIO HIPOTESIS”

Se desglosará los costes que se tomaron en cuenta para los cálculos del χ^2 de la hipótesis.

Debido a que nuestro ejército opera de manera ternaria, se va desglosando de a 3. En el caso de estudio nos basamos en el batallón de policía militar “Estaban Arce” el cual al ser un batallón contiene 3 compañías, y cada compañía compuesta por 81 soldados.

Entonces tenemos lo siguiente:

$$reg = 3 \text{ comp} * 81 \text{ soldados}$$

$$reg = 243 \text{ soldados}$$

Ahora recapitulando lo mencionado en el Anexo A se menciona que por soldado se requieren un total de 33 cartuchos para realizar la práctica de tiro y como se mencionó dentro de los antecedentes de este documento, sabemos que el costo de cada munición de calibre 7.62mm es de 5.14 Bs y el costo de cada munición de calibre 5.56 es de 5 Bs.

Tomando en cuenta todos estos parámetros se tiene los siguientes costos por cada entrenamiento.

$$Total \text{ Cartuchos} = 243(u) * 33(u)$$

$$Total \text{ Cartuchos} = 8019(u)$$

u = unidades.

Para el entrenamiento con munición 7.62mm se tiene un costo de:

$$Costo = 8019 \text{ cartuchos} * 5.14 \text{ Bs}$$

$$Costo = 41,217.66 \text{ Bs}$$

Para el entrenamiento con munición 5.56mm se tiene un costo de:

$$Costo = 8019 \text{ cartuchos} * 5 \text{ Bs}$$

$$Costo = 40,095 \text{ Bs}$$

Entonces se tiene que para cada entrenamiento de tiro se realiza un gasto aproximado de 40095 Bs para la munición de 5.56mm y un gasto de 41217.66 Bs para el entrenamiento con munición 7.62mm.

Para el cálculo del costo del entrenamiento con simulador se tomarán en cuenta los siguientes parámetros.

Costo del kWh en Cochabamba, Bolivia es de 1.04Bs y el W consumido por la maquina propuesta es de 650W y pensando que el equipo este encendido durante 16 horas todos los días se tiene lo siguiente.

$$kWhEquipo = 650 W * 16 horas$$

$$kWEquipoDia = 10.4$$

$$kWEquipoMes = 10.4 * 30Dias$$

$$kWEquipoMes = 312kW$$

Entonces se tiene que el costo:

$$CostoMesSimulador = 312 * 1.04$$

$$CostoMesSimulador = 324.48Bs$$

ANEXO “F”
SCRIPT PARA LOS
PARAMETROS DEL
ARMA



ANEXO “F” SCRIPT PARA LOS PARAMETROS DEL ARMA

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Valve.VR;
using Valve.VR.InteractionSystem;

public class Gun : MonoBehaviour
{
    public bool automatic = true;
    public float shootdelay = 0.1f;
    public SteamVR_Action_Boolean fireAction;
    public GameObject bullet;
    public Transform barrelPivot;
    public float shootingSpeed = 1;
    public GameObject muzzleFlash;

    private Animator animator;
    private Interactable interactable;
    private float lastShoot = 0;
    private float counter = 0;

    private float vientoForce;

    public GameObject magazine;
    public int bulletsCount;
    public bool whitoutAmmo;

    public SteamVR_Input_Sources leftHandle;
    // Start is called before the first frame update

    public int kills;
    int numberBulletsShoot = 0;
    void Start()
    {
        animator = GetComponent<Animator>();
        muzzleFlash.SetActive(false);
        interactable = GetComponent<Interactable>();
        vientoForce = Random.Range(9.9f, 22f) * 0.015f;
        bulletsCount = 30;
    }

    void Update()
    {
        if (SteamVR_Input.GetStateDown("X", leftHandle)) {
            print("press X");
            bulletsCount = 30;
            whitoutAmmo = false;
            magazine.SetActive(true);
        }
    }

    // Update is called once per frame
    void LateUpdate()
```

```

{
    //check if grabbed
    if(interactable.attachedToHand != null)
    {
        //get the hand source
        SteamVR_Input_Sources source =
interactable.attachedToHand.handType;

        //check button is down
        if(!automatic && fireAction[source].stateDown &&
!whitoutAmmo)
        {
            Fire();
            lastShoot = 0;
        }
        else if(automatic && lastShoot>shootdelay &&
fireAction[source].state && !whitoutAmmo)
        {
            Fire();
            lastShoot = 0;
        }
    }

    lastShoot += Time.deltaTime;
}

void Fire()
{
    Debug.Log("Fire");
    print("Fire");
    Rigidbody bulletrb = Instantiate(bullet,
barrelPivot.position,barrelPivot.rotation).GetComponent<Rigidbody>();
    bulletrb.velocity = barrelPivot.forward * shootingSpeed;
    bulletrb.AddForce(new Vector3(0, vientoForce,
0),ForceMode.Acceleration);
    bulletrb.GetComponent<ApplyDamage>().gun = this;
    muzzleFlash.SetActive(true);
    bulletsCount--;
    numberBulletsShoot++;
    if (bulletsCount <= 0) {
        whitoutAmmo = true;
        magazine.SetActive(false);
    }
}

public void ActiveMagazine() {
    magazine.SetActive(true);
    bulletsCount = 30;
    whitoutAmmo = false;
}

public void saveScore(float time)
{
    float precision = kills / numberBulletsShoot;
    PlayerPrefs.SetInt("kills", kills);
}

```



```
    PlayerPrefs.SetFloat("precision", precision);  
    PlayerPrefs.SetFloat("time", time);  
    PlayerPrefs.SetInt("saveScore", 1);  
}  
}
```

ANEXO “G”
ARCHIVO
SQLITECONTROLLER



ANEXO “G” ARCHIVO SQLITECONTROLLER

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Data;
using Mono.Data.Sqlite;
using System.IO;
using UnityEngine.UI;

public class SqliteController : MonoBehaviour
{
    IDbConnection dbcon;
    public GameObject cell;
    public GameObject parentContent;
    void Start() {
        string connection = "URI=file:" + Application.dataPath + "/" +
"leaderboard.db";
        // Open connection
        dbcon = new SqliteConnection(connection);
        dbcon.Open();
        if(PlayerPrefs.GetInt("saveScore" ) == 1)
        {
            addScore(PlayerPrefs.GetInt("kills"),
PlayerPrefs.GetFloat("precision"),
PlayerPrefs.GetFloat("time"), "Crisner");
            PlayerPrefs.SetInt("saveScore", 0);
        }
        List<Puntuacion> puntuaciones= getLeaderBoard();
        foreach (var i in puntuaciones) {
            GameObject obj = Instantiate(cell, Vector3.zero,
Quaternion.identity);

            obj.transform.SetParent(parentContent.transform);
            obj.transform.localScale = Vector3.one;

            obj.transform.GetChild(0).GetComponent<Text>().text =
i.id.ToString();

            obj = Instantiate(cell, Vector3.zero,
Quaternion.identity);

            obj.transform.SetParent(parentContent.transform);
            obj.transform.localScale = Vector3.one;

            obj.transform.GetChild(0).GetComponent<Text>().text =
i.userId.ToString();

            obj = Instantiate(cell, Vector3.zero,
Quaternion.identity);

            obj.transform.SetParent(parentContent.transform);
            obj.transform.localScale = Vector3.one;
```

```

        obj.transform.GetChild(0).GetComponent<Text>().text =
i.kills.ToString();

        obj = Instantiate(cell, Vector3.zero,
Quaternion.identity);

        obj.transform.SetParent(parentContent.transform);
        obj.transform.localScale = Vector3.one;

        obj.transform.GetChild(0).GetComponent<Text>().text =
i.precision.ToString();

        obj = Instantiate(cell, Vector3.zero,
Quaternion.identity);
        obj.transform.SetParent(parentContent.transform);
        obj.transform.localScale = Vector3.one;

        obj.transform.GetChild(0).GetComponent<Text>().text =
i.tiempo.ToString();
    }
}

public int searchUser(string name) {
    // Read and print all values in table
    IDbCommand cmdnd_read = dbcon.CreateCommand();
    IDataReader reader;
    string query = "SELECT * FROM usuarios WHERE nombre LIKE
\'" + name + "\'";
    print(query);
    cmdnd_read.CommandText = query;
    reader = cmdnd_read.ExecuteReader();
    int idUser = 0;
    while (reader.Read()) {
        Debug.Log("id: " + reader[0].ToString());
        Debug.Log("Nombre: " + reader[1].ToString());
        idUser = int.Parse(reader[0].ToString());
        PlayerPrefs.SetInt("userId", idUser);
    }
    if (idUser == 0) {
        idUser=createUser(name);
    }
    return idUser;
}

public int createUser(string name) {
    IDbCommand cmdnd = dbcon.CreateCommand();
    cmdnd.CommandText = "INSERT INTO usuarios (userId, nombre,
fechaCreacion) VALUES (NULL,\'" + name + "\',2020)";
    cmdnd.ExecuteNonQuery();
    return searchUser(name);
}

public void addScore(int kills, float precisionF, float tiempo,
string name) {

```

```

        int userID = searchUser(name);
        TimeSpan time =
TimeSpan.FromSeconds(Convert.ToInt32(tiempo));
        string tiempoString = time.ToString(@"mm\:ss");
        int precision = Convert.ToInt32(precisionF);
        // Insert values in table
        IDbCommand cmdnd = dbcon.CreateCommand();
        cmdnd.CommandText = "INSERT INTO puntuacion (puntuacionId,
kills, precision, tiempo, userId) VALUES
(NULL, "+kills+", "+precision+", \"'+tiempoString+'\", "+userID+)";
        cmdnd.ExecuteNonQuery();
    }

    public void closeDB() {
        // Close connection
        dbcon.Close();
    }

    public List<Puntuacion> getLeaderBoard() {
        IDbCommand cmdnd_read = dbcon.CreateCommand();
        IDataReader reader;
        string query = "SELECT * FROM puntuacion ORDER BY
precision DESC";
        cmdnd_read.CommandText = query;
        reader = cmdnd_read.ExecuteReader();
        List<Puntuacion> puntuaciones = new List<Puntuacion>();
        while (reader.Read()) {
            puntuaciones.Add(new
Puntuacion(reader[0].ToString(), reader[1].ToString(),
reader[2].ToString(), reader[3].ToString(), reader[4].ToString()));
        }
        return puntuaciones;
    }

    public void CrearArchivoCSV()
    {

        string ruta =
@"C:\Users\MrFro\OneDrive\Escritorio\ReportesSimulador" +
DateTime.Now.ToString() + ".csv";

        //El archivo existe? lo BORRAMOS
        if (File.Exists(ruta))
        {
            File.Delete(ruta);
        }
        IDbCommand cmdnd_read = dbcon.CreateCommand();
        IDataReader reader;
        string query = "SELECT * FROM puntuacion WHERE userId =
"+PlayerPrefs.GetInt("userId");
        cmdnd_read.CommandText = query;
        reader = cmdnd_read.ExecuteReader();
        List<Puntuacion> puntuaciones = new List<Puntuacion>();
        while (reader.Read())
        {

```

```

        puntuaciones.Add(new
Puntuacion(reader[0].ToString(), reader[1].ToString(),
reader[2].ToString(), reader[3].ToString(), reader[4].ToString()));
    }
    var sr = File.CreateText(ruta);
    string datosCSV="";
    datosCSV += puntuaciones[0].userId.ToString() + ", \n";
    datosCSV += "\n," + "\n," + "\n";
    datosCSV += "precision," + "aciertos," + "tiempos," +
"\n";

    foreach (var i in puntuaciones)
    {
        datosCSV += i.precision.ToString() + "," +
i.kills.ToString() + "," + i.tiempo.ToString() + ", \n";
    }

    //Crear el archivo

    sr.WriteLine(datosCSV);

    //Dejar como sólo de lectura
    FileInfo fInfo = new FileInfo(ruta);
    fInfo.IsReadOnly = true;

    //Cerrar
    sr.Close();

    //Abrimos archivo recién creado
    Application.OpenURL(ruta);
}

}

public class Puntuacion {
    public int id;
    public int kills;
    public int precision;
    public string tiempo;
    public int userId;

    public Puntuacion(string id, string kills, string precision,
string tiempo, string userId) {
        this.id = int.Parse(id);
        this.kills = int.Parse(kills);
        this.precision = int.Parse(precision);
        this.tiempo = tiempo;
        this.userId = int.Parse(userId);
    }
}

```

ANEXO “H”
SCRIPT PARA
CREACIÓN DE TABLA
DINÁMICA



ANEXO “H” SCRIPT PARA CREACIÓN DE TABLA DINÁMICA

```
List<Puntuacion> puntuaciones= getLeaderBoard();
    foreach (var i in puntuaciones) {
        GameObject obj = Instantiate(cell, Vector3.zero,
Quaternion.identity);

        obj.transform.SetParent(parentContent.transform);
        obj.transform.localScale = Vector3.one;

        obj.transform.GetChild(0).GetComponent<Text>().text =
i.id.ToString();

        obj = Instantiate(cell, Vector3.zero,
Quaternion.identity);

        obj.transform.SetParent(parentContent.transform);
        obj.transform.localScale = Vector3.one;

        obj.transform.GetChild(0).GetComponent<Text>().text =
i.userId.ToString();

        obj = Instantiate(cell, Vector3.zero,
Quaternion.identity);

        obj.transform.SetParent(parentContent.transform);
        obj.transform.localScale = Vector3.one;

        obj.transform.GetChild(0).GetComponent<Text>().text =
i.kills.ToString();

        obj = Instantiate(cell, Vector3.zero,
Quaternion.identity);

        obj.transform.SetParent(parentContent.transform);
        obj.transform.localScale = Vector3.one;

        obj.transform.GetChild(0).GetComponent<Text>().text =
i.precision.ToString();

        obj = Instantiate(cell, Vector3.zero,
Quaternion.identity);
        obj.transform.SetParent(parentContent.transform);
        obj.transform.localScale = Vector3.one;

        obj.transform.GetChild(0).GetComponent<Text>().text =
i.tiempo.ToString();
    }
```


ANEXO “I”
SCRIPT PARA
GENERAR REPORTES



ANEXO "I" SCRIPT PARA GENERAR REPORTES

```
public void CrearArchivoCSV()
{
    string ruta =
@"C:\Users\MrFro\OneDrive\Escritorio\ReportesSimulador" +
DateTime.Now.ToString() + ".csv";

    //El archivo existe? lo BORRAMOS
    if (File.Exists(ruta))
    {
        File.Delete(ruta);
    }
    IDbCommand cmdnd_read = dbcon.CreateCommand();
    IDataReader reader;
    string query = "SELECT * FROM puntuacion WHERE userId =
"+PlayerPrefs.GetInt("userId");
    cmdnd_read.CommandText = query;
    reader = cmdnd_read.ExecuteReader();
    List<Puntuacion> puntuaciones = new List<Puntuacion>();
    while (reader.Read())
    {
        puntuaciones.Add(new
Puntuacion(reader[0].ToString(), reader[1].ToString(),
reader[2].ToString(), reader[3].ToString(), reader[4].ToString()));
    }
    var sr = File.CreateText(ruta);
    string datosCSV="";
    datosCSV += puntuaciones[0].userId.ToString() + ",\n";
    datosCSV += "\n," + "\n," + "\n";
    datosCSV += "precision," + "aciertos," + "tiempos," +
"\n";

    foreach (var i in puntuaciones)
    {
        datosCSV += i.precision.ToString() + "," +
i.kills.ToString() + "," + i.tiempo.ToString() + ",\n";
    }

    //Crear el archivo

    sr.WriteLine(datosCSV);

    //Dejar como sólo de lectura
    FileInfo fInfo = new FileInfo(ruta);
    fInfo.IsReadOnly = true;

    //Cerrar
    sr.Close();

    //Abrimos archivo recién creado
    Application.OpenURL(ruta);
}
```

ANEXO “J”

MODELOS 3D EXTRAS



ANEXO “J” MODELOS 3D EXTRAS

Modelo 3D soldado enemigo



Modelo 3D Munición 7.62mm



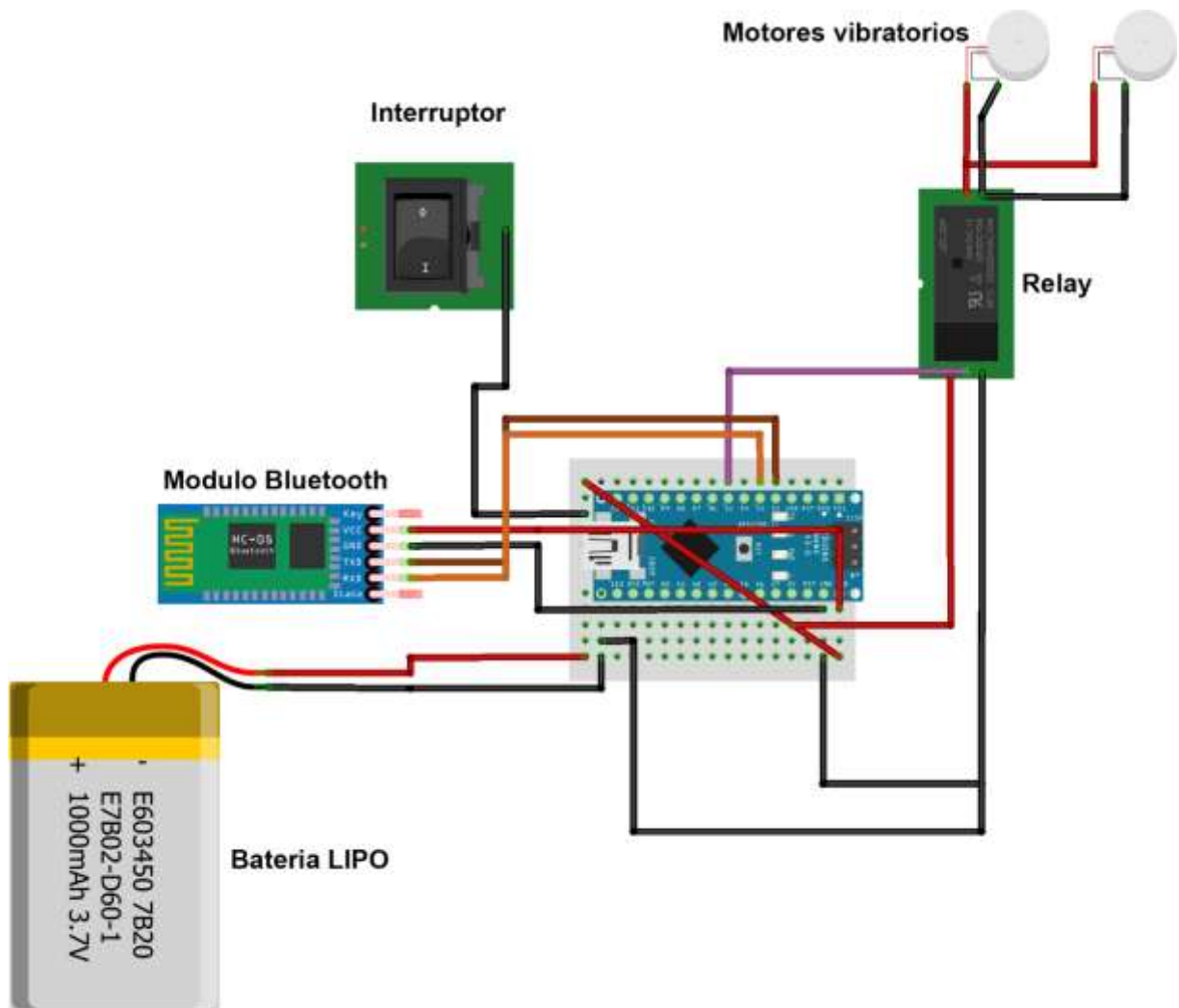
Modelo 3D Munición 5.56mm



ANEXO “K”
DIAGRAMA DEL
CIRCUITO E
IMPLEMENTACIÓN



ANEXO “K” DIAGRAMA DEL CIRCUITO E IMPLEMENTACIÓN





ANEXO “L”

CÓDIGO ARDUINO



ANEXO "L" CODIGO ARDUINO

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(2, 3); // RX, TX
char data;
void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ;
  }
  mySerial.begin(115200);
  pinMode(5, OUTPUT);
  digitalWrite(5, HIGH);
}

void loop() {
  if (mySerial.available()) {
    data=mySerial.read();
    if (data==112) {
      digitalWrite(5, HIGH);
      Serial.println("apagar");

    } else if (data==102) {
      digitalWrite(5, LOW);
      Serial.println("prender");
    }
  }
  /* if (Serial.available())
    mySerial.write(Serial.read()); */
  data = Serial.read();
  if (data==112) {
    digitalWrite(5, HIGH);
    Serial.println("apagar");

  } else if (data==102) {
    digitalWrite(5, LOW);
    Serial.println("prender");
  }
}
```