UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Materiales digitales para uso de mesas interactivas en terapias de rehabilitación neurológica

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

José Martín Arreola Manzo

DIRECTOR DE TESIS

Dr. Rodrigo Montúfar Chaveznava

Ciudad Universitaria, Cd. Mx., 2017

Agradecimientos

Investigación realizada gracias al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT) de la UNAM en el Proyecto “Desarrollo de Recursos Interactivos de Bajo Costo Para Neuro-Rehabilitación”, clave IT 102215.

Agradezco a la DGAPA-UNAM la beca recibida.

A María Isabel Heredía, especialista en neurorehabilitación por su guía en esta Investigación. A Gerardo Coello, a Ana María Escalante y a Yoás Saimon Ramírez, responsables del Laboratorio de Desarrollo de Aplicaciones Interactivas para la Neuro-rehabilitación en el Instituto de Fisiología Celular. Por su asesoría, tiempo y paciencia. A la Dra. Herminia Pasantes, promotora del laboratorio, por su interés en esta iniciativa.

Al Doctor Rodrigo Montúfar Chaveznava por su guía y paciencia a lo largo del desarrollo de este trabajo de tesis.

A los sinodales M.I. Juan Manuel Gomez Gonzales, M.C. Eduardo Espinosa Ávila, Ing. Luis Sergio Valencia Castro e Ing. José Roque Roman Guadarrama por su guía y paciencia en la revisión del presente trabajo de tesis.

Dedicatoria

Agradezco a mi Madre, a mi Padre y a mi Hermana por todo el apoyo brindado a lo largo de la carrera.

A mis amigos de la Facultad, Luis Enrique, Leonardo, Mario y Fernando, que me brindaron su amistad a lo largo de toda la carrera.

A mi tutora de la Facultad, quién me brindo su apoyo y guía durante toda la carrera.

A Yoás, ya que sin él no hubiera podido conocer la propuesta del tema del presente desarrollo de Tesis del Laboratorio de Desarrollo de Aplicaciones Interactivas para la Neuro-rehabilitación en el Instituto de Fisiología Celular.

Índice

1. Objetivos

1.1 Objetivo general

Desarrollar una herramienta de *software* de realidad virtual complementaria a las terapias de neurorehabilitación de pacientes para el reentrenamiento de la escritura, que utilice pantallas táctiles y el sensor *Kinect One* para PC como dispositivos de entrada.

1.2 Objetivos particulares

-Desarrollar un conjunto de herramientas de *software* gráficas que sirvan como complementos a las terapias convencionales de neurorehabilitación

-Realizar las adaptaciones necesarias para que las herramientas de *software* utilicen como dispositivos de entrada, en un primer caso el sensor *Kinect One*, y en el segundo caso una pantalla táctil.

-Realizar algunas pruebas con las herramientas de *software*, en primera instancia con voluntarios, para conocer las mejoras que pudiera requerir el sistema. Una vez el sistema se encuentre integrado con el sensor *Kinect One* y con una pantalla táctil.

-Obtener la versión final de las herramientas de *software* con base a lo obtenido en el objetivo anterior.

2. Planteamiento del problema

Se requiere desarrollar un conjunto de herramientas de *software* gráficas, que permitan al terapeuta utilizarlas para complementar los ejercicios de neurorehabilitación habituales. Para permitir una comunicación humano-máquina más natural o al menos accesible para los pacientes, las herramientas de *software* utilizarán como dispositivos de entrada, en un primer caso el sensor *Kinect One* y en el segundo caso una pantalla táctil.

La herramienta de *software* es un programa gráfico, por tanto es posible utilizar un motor gráfico para desarrollarlo. Además, hoy en día existe un gran soporte en línea para las interacciones *Kinect*-Computadora y Pantalla táctil-Computadora dentro de diferentes motores gráficos.

Además de los programas que se requerirán escribir para el funcionamiento de la herramienta, será necesario crear los modelos 3D pertinentes, utilizando algún *software* de modelado 3D.

3. Justificación

De acuerdo a datos estadísticos de la ONU, aproximadamente mil millones de personas en el mundo padecían algún desorden neurológico en 2007. Se sabe que, 50 millones sufrían de epilepsia, 62 millones de accidentes cerebrovasculares, 326 millones de migraña y 24 millones de la enfermedad de Alzheimer y otras demencias. **[26]**

Por su parte, la organización mundial de la salud, indicó en su informe de 2002 que aproximadamente cada año 15 millones de personas sufren de accidentes cerebrovasculares a nivel mundial. De estos, 5 millones mueren y 5 millones quedan permanentemente discapacitados. Más de 12.7 millones de accidentes cerebrovasculares se deben a la presión sanguínea alta. **[22]**

La organización mundial de la salud pronostica que para 2030, la cantidad de personas que sufrirán de accidente cerebrovascular y fallecerán por la misma causa serán poco más de 8 millones, de continuar la tendencia actual. **[10]**

En otras palabras, esta enfermedad estará presente en nuestra población y la de muchos países por muchos años más en el futuro, y si no podemos curarla o evitarla, entonces es necesario tratarla de la mejor manera posible para mejorar la calidad de vida del paciente tanto como sea posible.

Existen diferentes terapias neurológicas utilizadas en clínicas con un alto índice de efectividad al tratar diferentes padecimientos. Sin embargo, la tecnología ha evolucionado considerablemente, por lo cual es plausible que el terapeuta utilice herramientas adicionales, basadas en sistemas computarizados que complementan las terapias convencionales. Principalmente, con el fin de que la terapia sea más llamativa o lúdica.

Adicionalmente, el paciente podrá ver cómo van mejorando sus habilidades al observar los resultados obtenidos al finalizar cada ejercicio. De esta forma, el paciente y el terapeuta pueden analizar resultados numéricos sencillos.

4. Fundamentos teóricos

4.1 La neurorehabilitación

En los años 90, se produjeron enormes progresos en el conocimiento científico de esta rama de las ciencias, siendo crucial para las nuevas concepciones en el tratamiento de las enfermedades neurológicas. En esta época surge la neurorehabilitación para dar respuesta a este tipo de patologías. **[16]**

La neurorehabilitación es una disciplina médica integrada por un equipo interdisciplinario que incluye médicos neurólogos, fisiatras, terapistas físicos, del lenguaje, psicopedagogos, ocupacionales, del ejercicio, etc. Con una visión de tratamiento de acuerdo a los diferentes métodos clásicos y al ejercicio físico terapéutico, siendo este último el medio fundamental en el proceso de neurorehabilitación como el mayor potente estimulador del sistema nervioso. **[16]**

El objetivo de la neurorehabilitación es estimular al sistema nervioso para que forme nuevas conexiones neuronales. **[15]**

Las últimas investigaciones demuestran que el cerebro y el tejido nervioso pueden generar nuevos circuitos neuronales mediante su uso y potenciación. Esta capacidad se llama plasticidad neuronal. **[15]**

El ejercicio físico terapéutico es el medio más ampliamente usado en los tratamientos de neurorehabilitación, este juega un rol decisivo en todas las áreas de neurorehabilitación (terapia física, del lenguaje, ocupacional e incluso psicológica), siendo determinante su aplicación para la restauración o mejora de la función dañada o pérdida por daño al sistema nervioso central o periférico. **[16]**

4.2 Objetivos y fundamentos de la neurorehabilitación

El objetivo de la neurorehabilitación es el de ayudar al paciente a recuperar el máximo nivel posible de funcionalidad e independencia y a mejorar su calidad de vida general tanto en el aspecto físico como en los aspectos psicológico y social. **[16]**

La práctica y la repetición son las claves en los que se apoya la neurorehabilitación, teniendo como sustrato la neuroplasticidad. La práctica y la repetición de los ejercicios en patrones de movimiento normales son los dos principios en los que debe basarse la rehabilitación de trastornos del sistema nervioso, dado que al fin y al cabo, lo que se necesita cuando uno olvida lo aprendido, por lesiones del sistema nerviosos u otros, es volver a aprenderlo. **[16]**

Estas teorías parten de la base que el cerebro lesionado continúa teniendo la capacidad de aprender. El aprendizaje tiene importantes implicaciones que permiten que el cerebro se reorganice en función de la información que le es suministrada, a través de ejercicios activos y estímulos. **[16]**

A través de la experiencia en la rehabilitación en pacientes con lesiones neurológicas, se ha demostrado que a través de la ejercitación motora, se puede recuperar parcial o totalmente las funciones alteradas mediante procedimientos de rehabilitación, a través de la ejercitación activa y la repetición de patrones musculares de movimiento normales logrando instaurar nuevos patrones de movimiento, modificando funcional y estructuralmente al cerebro. Depende en gran medida en esta recuperación el tipo de lesión, la etiología y el tiempo desde que se produjo la lesión. **[16]**

4.3 Sistema nervioso

Se mencionan brevemente las partes que forman este sistema y sus funciones principales.

Sistema nervioso central

-Cerebro: el cerebro es un órgano que forma parte del sistema nervioso central, localizado dentro del cráneo y cerca de los órganos sensoriales como por ejemplo los ojos. El cerebro está formado principalmente por neuronas y células gliales. Existen células gliales de distintos tipos y realizan diferentes funciones críticas tales como soporte estructural, soporte metabólico, aislamiento, etc. En el caso de las neuronas, son células que se caracterizan por la capacidad de enviar señales que permiten la comunicación entre ellas. Esto lo hacen por medio de los axones, una extensión del cuerpo celular. Cada axón puede comunicarse con hasta miles de células por medio del mecanismo de la sinapsis. La función de la sinapsis puede ser excitatoria (si se excita una célula objetivo), o inhibitoria. **[5]**

-Médula espinal: en conjunto con el cerebro, forma el sistema nervioso central y se encarga de la transmisión de señales neuronales entre el cerebro y el resto del cuerpo. También actúa en sentido inverso, transmitiendo señales desde el cuerpo hacia el cerebro. Además contiene ciertos circuitos neuronales que pueden controlar de manera independiente numerosos reflejos. **[5]**

-Hipotálamo: es una parte del cerebro que se encarga de funciones tales como ciertos procesos metabólicos y otras actividades del sistema nervioso autónomo. Sintetiza y secreta ciertas hormonas, denominadas hipotalámicas que a su vez estimulan o inhiben la secreción de hormonas pituitarias. El hipotálamo controla la tempetarua corporal, el hambre, ciertos aspectos de las conductas parentales y de apego, fatiga, sueño, etc. **[5]**

-Tálamo: parte del cerebro que se encarga principalmente de la retransmisión de señales motoras y sensoriales hacia la corteza cerebral. También se encarga de ciertos aspectos de la regulación del estado de vigilia, sueño y alerta. Se encuentra conectado funcionalmente al hipocampo y es crucial para la memoria espacial y sensorial. **[5]**

-Cerebelo: es una región del cerebro que juega un rol curcial en el control motor. Aunque tiene otras funciones no del todo definidas, principalmente se considera que se encarga de funciones relacionadas con el movimiento del cuerpo. Se observa que en personas y animales con disfunción cerebelar, estas son capaces de ejecutar movimientos pero con precisión reducida, generalmente ejecutándolos de manera errática o con un ritmo incorrecto. De este modo, juega un rol particular para el control de la coordinación motriz fina. **[5]**

-Hipocampo: esta estructura del cerebro, se encarga principalmente de aspectos relacionados con la memoria de corto y largo plazo, así como de la memoria espacial. **[5]**

Sistema nervioso periférico

El sistema nervioso periférico está formado por nervios y glangios fuera del cerebro y la médula espinal. Su función principal es la conexión entre el sistema nervioso central, desde la médula espinal, hacia y desde las distintas partes del cuerpo. **[17]**

A su vez está dividido en sistema nervioso somático, sensorial y autónomo. El somático, se encuentra bajo control voluntario y transmite señales desde el cerebro hasta órganos como los músculos esqueléticos. El sensorial, transmite señales desde los órganos sensoriales como la lengua o los dedos hacia la médula espinal. El autónomo, es un sistema autoregulado que actúa en las funciones de órganos para los cuales se tiene muy poco o nulo control voluntario, tales como aquellos que forman parte del sistema digestivo. **[17]**

4.4 Coordinación motriz fina

El control de la motricidad fina es la coordinación de músculos, huesos y nervios para producir movimientos pequeños y precisos. **[6]**

Lo opuesto al control de la motricidad fina es el control de la motricidad gruesa. Un ejemplo de control de la motricidad gruesa es agitar los brazos al saludar. **[6]**

Tanto los problemas del cerebro, la médula espinal, como de los nervios periféricos (los nervios que están fuera del cerebro y de la médula espinal), los músculos o las articulaciones pueden afectar el control de la motricidad fina. **[6]**

4.5 ¿Por qué es necesario un reentrenamiento de la escritura?

Mientras que para una persona sana, la escritura pudiera parecer una actividad común e incluso sencilla, se considera que es una de las habilidades que involucran coordinación motriz fina más compleja. Otras actividades tales como dibujar, recortar papel, atar nudos e incluso utilizar cubiertos para comer, también requieren del uso de la coordinación motriz fina.

Cuando una persona sufre un accidente cerebrovascular (padecimiento que se explica más adelante), diferentes habilidades se pueden ver afectadas en menor o mayor grado, dependiendo del área dañada. Ya sea que el paciente presente problemas de espasticidad, perdida de la coordinación ojo-mano o de la motriz fina, será poco probable que en este punto sea capaz de ejecutar algunas tareas cotidianas y en especial de escribir. Es entonces cuando decimos que el paciente requiere un reentrenamiento de la escritura.

Como es natural, antes de empezar a reentrenar al paciente en la escritura, deberá recibir tratamientos de neurorehabilitación para recuperar sus habilidades básicas, tanto la capacidad de cerrar y abrir la mano, como la coordinación ojo-mano o específicamente la motriz fina, y es hasta ese punto cuando el paciente estará listo para recibir un reentrenamiento de la escritura.

4.6 Accidente cerebrovascular

El ictus o accidente cerebrovascular se origina por un trastorno brusco de la circulación sanguínea. Esta alteración del flujo sanguíneo cerebral produce que una región determinada del cerebro deje de funcionar rápidamente. Motivo por el cual, la palabra “ictus” que proviene del latín, significa “golpe”, haciendo referencia a la brusquedad con que aparecen los síntomas. Éstos dependen de la zona del cerebro que se vea afectada, ya que como es sabido, cada región cerebral se encarga de una función concreta, por lo que los síntomas pueden consistir en una pérdida de fuerza o sensibilidad en una mitad del cuerpo, dificultad para hablar o comprender el lenguaje, pérdida de visión o dificultad para caminar. **[2]**

Existen dos tipos principales de ictus, que se diferencian por el mecanismo de la alteración vascular. El más frecuente es el isquémico, que representa el 80 % de los casos. En este caso se produce una oclusión de un vaso sanguíneo cerebral, que origina una disminución o ausencia de aporte de sangre a una región del cerebro. Cuando una zona del cerebro no recibe la sangre suficiente deja de funcionar y aparecen manifestaciones clínicas como las anteriormente mencionadas. Si el flujo sanguíneo se reestablece rápidamente cuando las células del cerebro todavía no han sido dañadas,

los síntomas pueden resolverse sin dejar ningún tipo de secuela. En este caso hablamos de “ataque isquémico transitorio”. Si el flujo sanguíneo no se restaura a tiempo, se produce una lesión cerebral definitiva derivada de la muerte de los distintos tipos de células que hay en el cerebro, lo que denominamos “infarto cerebral”. El otro tipo de ictus es el hemorrágico, o “hemorragia cerebral”, que representa el 20% restante de todos los ictus. La hemorragia cerebral se produce cuando se rompe un vaso sanguíneo dentro del cerebro. La rotura de un vaso sanguíneo hace que el tejido cerebral se inunde de sangre, alterando la función de las células y pudiendo ocasionar un daño reversible. **[2]**

Algunos ictus isquémicos pueden producirse por una lesión directa sobre una arteria y otros por una alteración en el corazón que favorece la liberación de trombos hacia la circulación cerebral. De este modo según su causa, clasificamos los ictus isquémicos en aterotrombóticos, lacunares, cardioembólicos o de causa inhabitual; en los casos en los que la causa no está completamente aclarada hablamos de ictus de origen indeterminado. **[2]**

Existen diferentes causas que pueden originar una hemorragia cerebral. La más frecuente es la hipertensión arterial, que condiciona alteraciones en la pared de los vasos que pueden provocar su rotura. La hipertensión arterial crónica produce una afectación de las pequeñas arterias perforantes que condiciona una debilidad de su pared. Esta debilidad favorece la formación de pequeñas dilataciones (microaneurismas), las cuales pueden romperse con facilidad. **[2]**

Otra causa frecuente de hemorragias cerebrales son los trastornos de coagulación, de los cuales algunos son congénitos y están presentes desde el nacimiento, y otros son adquiridos. La administración de fármacos que produzcan una alteración de la coagulación o de las plaquetas también es una causa frecuente de hemorragias cerebrales. **[2]**

En los países occidentales, el ictus cerebral es la principal causa de discapacidad física grave en el adulto. A los 6 meses del ictus, aproximadamente un 20-25% de los supervivientes continúan siendo incapaces de caminar sin asistencia física y más de un 60 % no pueden incorporar la mano afecta a la realización de las actividades de la vida diaria. **[2]**

El ictus provoca una gran variedad de déficits y discapacidad en el paciente. **[2]**

La *American Heart Association-Stroke Outcome Classification* sistematiza los déficits neurológicos en seis áreas: motora, sensitiva, visual, de lenguaje o comunicación, cognitiva o del intelecto, y emocional. **[2]**

El déficit motor suele ser unilateral. La hemiparesia o hemiplejia constituye el déficit más prevalente tras un ictus. El déficit motor es la causa principal de discapacidad física. **[2]**

4.7 Neuroplasticidad

Como ya se ha indicado, la neuroplasticidad o plasticidad neuronal es la habilidad del tejido nervioso para autorreorganizarse a partir de la remodelación funcional, a corto o a largo plazo, de las conexiones neuronales o sinapsis. Los fenómenos de activación o depresión a largo plazo se deben a cambios celulares y moleculares implicados en la neuroplasticidad. **[2]**

4.8 Espasticidad

Se refiere a músculos tensos y rígidos. También se puede llamar tensión inusual o aumento del tono muscular. Los reflejos (por ejemplo, un reflejo rotuliano) son más fuertes o exagerados. La afección puede interferir con la actividad de caminar, el movimiento o el habla. **[8]**

Generalmente es causada por daños al cerebro en áreas que controlen dichos músculos o bien daño a nervios que van del cerebro a la médula espinal. **[8]**

Los síntomas de espasticidad abarcan: **[8]**

*Postura anormal.

*Llevar los hombros, los brazos, la muñeca y los dedos de las manos a un ángulo anormal debido a la rigidez muscular.

*Reflejos tendinosos profundos y exagerados (el reflejo rotuliano y otros reflejos).

*Movimientos espasmódicos repetitivos (clono), especialmente al tocarlo o moverlo.

*Tijereteo (cruce de piernas como se cerrarían las puntas de unas tijeras).

La espasticidad tiene múltiples causas, entre las que están: **[8]**

-Adrenoleucodistrofia

-Daño cerebral causado por falta de oxígeno, como puede ocurrir con asfixia o ahogamiento inminente

-Parálisis cerebral

-Traumatismo craneal

-Esclerosis múltiple

-Enfermedad neurodegenerativa (afección que daña el cerebro y el sistema nervioso con el tiempo)

-Fenilcetonuria

-Lesión de la médula espinal

-Accidente cerebrovascular

4.9 Coordinación ojo-mano

La coordinación ojo-mano se puede definir como la habilidad que nos permite realizar actividades en las que utilizamos simultáneamente los ojos y las manos. Utilizamos los ojos para dirigir la atención y las manos para ejecutar una tarea determinada. La coordinación ojo-mano es una habilidad cognitiva compleja, ya que debe guiar los movimientos de nuestra mano de acuerdo a los estímulos visuales y de retroalimentación. **[7]**

La coordinación ojo-mano es básica para el desarrollo y ejecución de la lectoescritura, pero también la utilizamos diariamente en infinidad de actividades de nuestra vida diaria como por ejemplo, cuando cocinamos, tecleamos en la computadora, etc. También, cuando conducimos, utilizamos nuestra coordinación ojo-manual de forma ininterrumpida: Basándonos en la información sobre el trayecto que percibimos a través de los ojos, las manos actúan constantemente sobre el volante. **[7]**

4.10 Programas gráficos

Cuando se habla de “programas gráficos”, se está haciendo referencia a programas de computadora que se caracterizan por utilizar la aceleradora gráfica y la CPU del sistema para mostrar en un dispositivo de despliegue, gráficos generados en tiempo real que son la representación visual de un entorno virtual 3D y que normalmente reaccionan acorde a los datos que se reciben a través de los dispositivos de entrada. Esto se logra con el uso de uno o varios lenguajes de programación (usualmente de alto nivel) en conjunto con el uso de una API gráfica específica (Direct3D u Open GL) y de ser necesario uno o más *shaders* en uno o más lenguajes de programación. Es importante mencionar que hoy en día casi todos los sistemas operativos utilizan en menor o mayor medida las diferentes técnicas de los gráficos por computadora para la interfaz de usuario del sistema operativo. Lo cual explica el gran auge de las aceleradoras gráficas integradas ya dentro del paquete de la CPU, puesto que sin este dispositivo el sistema operativo no podría funcionar correctamente o al menos no su interfaz de usuario. La razón de ser de un programa gráfico es que en general, las personas encuentran más llamativa la

información altamente visual que simplemente un conjunto de letras sobre el dispositivo de despliegue. Es por esto que prácticamente cualquier interfaz de usuario dirigida al usuario final utiliza indicaciones visuales para todo lo que pueda manipularse.

La tecnología que permite los gráficos por computadora, ha estado en constante desarrollo desde las primeras computadoras que contaban con pantallas verdosas de baja resolución, hasta nuestros días. Esto no solo incrementó la calidad visual de los gráficos por computadora sino que además se plantearon metas a largo plazo que van más allá de mostrar simples líneas de comandos. Gracias al avance de la tecnología, hoy uno de los objetivos principales de los gráficos por computadora es que en esencia, sean lo más parecidos a la realidad posible. Para lograr esto se han estado desarrollando y mejorando diferentes algoritmos que permiten simular el mundo real en apariencia dentro de un sistema computarizado.

De esta forma, han surgido un sin fin de técnicas más avanzadas que permiten simular en apariencia el mundo real con un grado alto de similitud. Entre las técnicas que existen, se tienen las técnicas de iluminación, de sombras, de oclusión, de animación, etc. Evidentemente, conforme las técnicas se vuelven más complejas y al mismo tiempo permiten una similitud mayor en apariencia con el mundo real, las exigencias del *hardware* para que una computadora pueda ejecutar un programa gráfico de esta naturaleza, se incrementan cada vez más. Afortunadamente, los principales fabricantes tanto de aceleradoras gráficas, como de CPU, memorias RAM, tarjetas madre, tarjetas de red y discos duros, están continuamente mejorando el desempeño de todos sus productos, sin mencionar que se están comercializando productos de esta naturaleza que cuentan con características muy particulares que los hacen perfectos para un sistema que vaya a utilizarse para gráficos por computadora. Esto se debe en parte a la mayor demanda de los programas gráficos, en especial los de Realidad Virtual, que no solo se están utilizando con fines de entretenimiento, sino que además se están utilizando en aplicaciones más serias que van desde los museos virtuales, la reconstrucción de estructuras arqueológicas relevantes dentro de una simulación, hasta el entrenamiento militar y médico. Por otro lado el auge del *Gaming* en PC, podríamos decir es la razón principal por la que el *hardware* especializado para gráficos por computadora se comercializa activamente en nuestros días.

4.11 Realidad virtual

La realidad virtual (o RV) se podría definir como un sistema informático que genera en tiempo real representaciones de la realidad, que no son más que proyecciones ya que se trata de una realidad perceptiva sin ningún soporte físico, en el sentido de que el mundo observado no existe físicamente. **[19]**

La simulación que hace la realidad virtual se puede referir a escenas virtuales, creando un mundo virtual que sólo existe en el ordenador de lugares u objetos que existen en la realidad. También permite capturar la voluntad implícita del usuario en sus movimientos naturales procesándolos en el mundo virtual que estamos generando, proyectando en el mundo virtual movimientos reales. **[19]**

4.12 Realidad aumentada

La Realidad Aumentada (RA) consiste en sobreponer objetos o animaciones generadas por computadora sobre la imagen en tiempo real que recoge una cámara, generalmente del tipo *web* o de un teléfono celular. **[18]**

De esta manera podemos "aumentar" en la pantalla, la realidad que mira la cámara con los elementos de una realidad virtual "Es el entorno real mezclado con lo virtual". **[18]**

A diferencia de la realidad virtual, la RA es una tecnología que complementa la percepción e interacción con el mundo real y permite al usuario estar en un entorno aumentado con información generada por una computadora. **[18]**

Los elementos básicos que conforman un sistema de RA son los siguientes: un monitor, una cámara, el *software* ex profeso y un marcador. El marcador es un símbolo escrito o impreso sobre objetos determinados, que puede ser un código de barras de cualquier producto, un código QR o bien la superficie de monumentos o edificios. **[18]**

La cámara se encarga de captar esos símbolos y transferirlos al *software*. Éste interpreta los datos de los marcadores captados por la cámara y los convierte en todo tipo de información: Texto, imágenes fijas, video en 3D o sonido. **[18]**

4.13 Aplicaciones

-Aplicaciones en general:

Las aplicaciones que en la actualidad encontramos de la realidad virtual a actividades de la vida cotidiana son muchas y diversas. Hay que destacar: la reconstrucción de la herencia cultural, la medicina, la simulación de multitudes y la sensación de presencia. **[19]**

La reconstrucción de la herencia cultural consiste en la recuperación a través de la simulación de piezas únicas de la antigüedad que han sido destruidas o se encuentran degradadas. En algunas, a partir de unos pocos restos se pueden simular piezas enteras. Además, la realidad virtual permite mostrar la pieza en perfecto estado en diversos lugares del mundo a la vez, e incluso permite crear museos enteros con piezas virtuales. **[19]**

La aplicación en la medicina la encontramos en la simulación virtual del cuerpo humano. A partir de imágenes de nuestro cuerpo, se puede hacer la recreación en 3D del paciente, cosa que facilita la elaboración de un diagnóstico, o la simulación de operaciones en caso que sea necesario. **[19]**

La simulación de multitudes consiste en la simulación del comportamiento de grandes cantidades de personas. Sin requerir la presencia de gente, se puede simular el comportamiento de éstas en situaciones específicas como la evacuación de un edificio. **[19]**

También, hay que destacar la aplicación de la realidad virtual en el campo de la presencia, simulando situaciones para inducir comportamientos en los individuos para aplicaciones como: tratar fobias, ansiedad social, estudios de violencia o resolución de conflictos. **[19]**

Finalmente, además de las finalidades tecnológicas, últimamente la realidad virtual ha llegado a juegos online en los cuales las personas simulan una segunda vida en un mundo virtual, en lo que el realismo conseguido provoca que los individuos durante un tiempo sean otra persona. **[19]**

-Aplicaciones concretas:

*En Deportes: De acuerdo al artículo “Terapia Visual como método de mejora de la coordinación ojo-mano y el tiempo de reacción visual de un tenista” del Instituto de Salud Visual, se utilizó en un sujeto de prueba una herramienta gráfica. Con el fin de mejorar las habilidades de un tenista veterano a través de la realización de diferentes ejercicios basados en programas gráficos. Básicamente, el sistema utilizado consistió en un

programa gráfico conectado a una pantalla táctil de 50 pulgadas. A diferencia del presente desarrollo de tesis, el objetivo no era rehabilitar a un paciente, sino simplemente mejorar las habilidades ya existentes en un tenista amateur con el fin de ser más competitivo en su deporte preferido. **[24]**

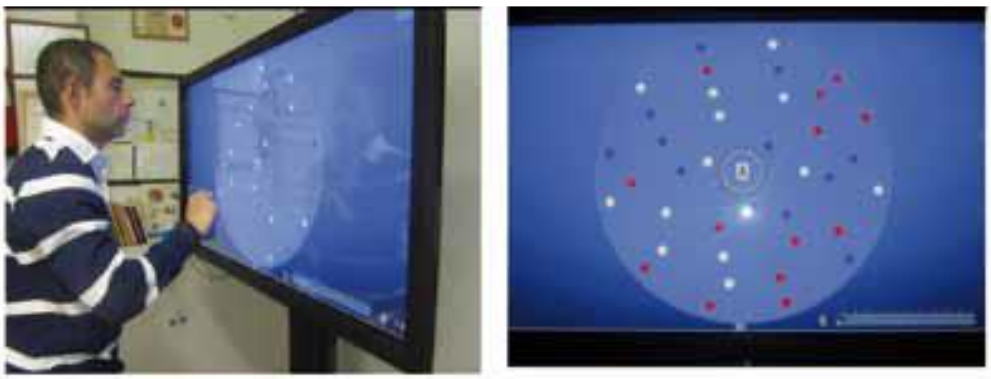


Figura 4.11.1 Herramienta gráfica del artículo mencionado

*****En la milicia: Probablemente un caso real muy conocido, es la aplicación de realidad virtual que permite a soldados de los Estados Unidos, mostrada en la página web de la Sociedad de Realidad Virtual (VRS o *Virtual Reality Society*), realizar entrenamiento del descenso en paracaídas. En este sistema se utilizan gafas de realidad virtual y algunos dispositivos mecánicos (como las correas que se observan en la figura 4.11.2) para simular el descenso en paracaídas. **[25]**

Figura 4.11.2 Soldado utilizando un sistema de entrenamiento militar basado en realidad virtual

*En la educación: en este caso podemos mencionar diferentes aplicaciones desarrolladas en la dependencia denominada DGTIC, específicamente en el Departamento de Visualización y Realidad Virtual, tanto para dispositivos móviles como para computadoras de escritorio. **[4]**

Algunas de ellas son: **[4]**

-“Microscopio electrónico virtual”: un ambiente virtual donde se muestra el funcionamiento y estructura física de un microscopio electrónico.

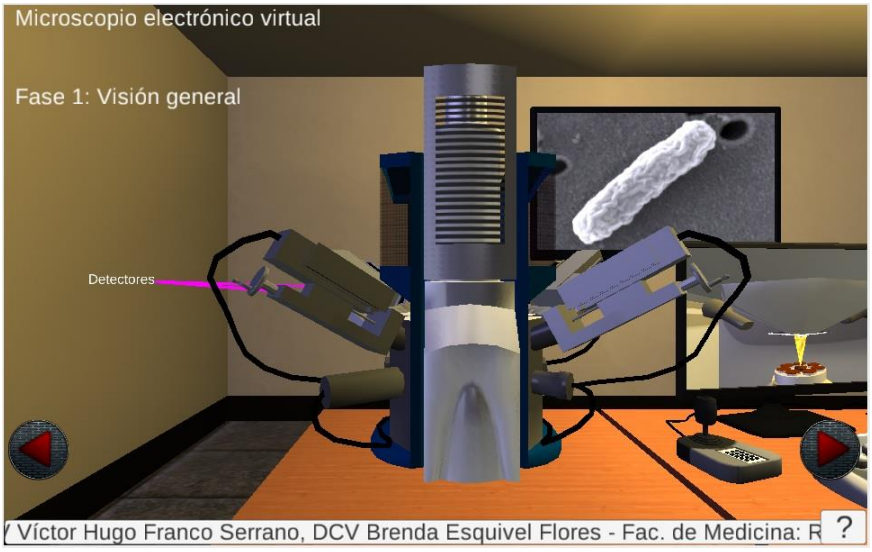


Figura 4.11.3 Captura del programa “Microscopio electrónico virtual”

-“Buque Oceanográfico Puma”: ambiente virtual donde se recrea el Barco Oceanográfico Puma, donde es posible observar las diferentes partes que conforman el barco de manera muy detallada.

Figura 4.11.4 Captura del programa “Buque Oceanográfico Puma”

*En Medicina: El 11 de Marzo de 2015, una compañía llamada *Medical Realities* anunció su primer producto en un evento denominado *London Wearable Technology Conference*. Este producto denominado *The Virtual Surgeon* (la cirugía virtual), es la primera cirugía capturada y editada para su visualización en un dispositivo de realidad virtual en el Reino Unido. **[14]**

Como se puede observar, existe una infinidad de aplicaciones para la Realidad Virtual en distintos campos de la ciencia. Puesto que como cualquier otro avance tecnológico, si se le da un buen uso puede traer beneficios tangibles a la población en general o bien a un grupo particular de individuos, sean pacientes, estudiantes, investigadores, etc.

5. Propuesta de solución

Se propone un programa gráfico en el que se simulen los ambientes virtuales necesarios para que el paciente ejecute diferentes ejercicios complementarios a su terapia habitual. Todos ellos tendrán que ser aprobados por la terapeuta responsable del laboratorio y estarán basados en los que se indiquen en algún libro también aprobado por la terapeuta o bien propuestos por ella misma.

El programa gráfico será desarrollado utilizando el motor gráfico Unity 3D en su versión 5.2.4 para el desarrollo de los ambientes virtuales.

Para el modelado 3D se utilizará el *software* 3DS Max 2016-Licencia de estudiante.

Para la integración de la pantalla táctil al sistema, se propone en principio utilizar los métodos nativos de C# en Unity 3D para manejar las entradas táctiles que lleguen al programa.

Para la integración del dispositivo *Kinect One* al sistema, se propone utilizar en principio el SDK de *Kinect For Windows* que provee Microsoft en su página oficial, así como el paquete de Unity 3D que también provee Microsoft.

6. Desarrollo de la solución

Una vez sustentada la viabilidad de la solución del problema, se decidió desarrollar la herramienta de *software* con base en ejercicios que se utilicen en neurorehabilitación para crear su contraparte en un ambiente virtual. Para el diseño de los ejercicios de la presente tesis, se tomaron como base aquellos mostrados en el libro Educar para Escribir. **[1]**

6.1 Elección de herramientas de software

Unity:

Desarrollador**:** Unity Technologies

Unity es un tipo de programa que en el área de gráficos por computadora, denominamos "Motor de juegos". Esto es, un programa que cuenta con un conjunto de herramientas de *software* que permiten el desarrollo ágil de programas gráficos ejecutables para uno o más sistemas operativos, empleando uno o más lenguajes de programación de alto nivel, tanto para programas para la CPU como para los shaders para la GPU.

Para desarrollar el programa gráfico se utilizó Unity 3D, se seleccionó este motor, puesto que Unity 3D cuenta con un gran soporte para sus propias herramientas y también para la integración de dispositivos de entrada como el *Kinect* y las pantallas táctiles, además en el laboratorio todos los proyectos se desarrollan con dicho motor y esto permite que se pueda modificar o reutilizar un mismo proyecto en otros. Se eligió desarrollar el sistema en lenguaje C# debido a que, dentro del rango de lenguajes posibles (JavaScript, C# y Boo), este lenguaje cuenta con mayor soporte y ejemplos prácticos. En el caso de JavaScript y Boo, estos lenguajes aún tienen ciertos problemas, en especial al trabajar en programas multiplataforma, especialmente por el uso de ciertas estructuras dinámicas.

Unity consta de dos partes fundamentales:

1) Programa principal Unity

Un simulador que se encarga de ejecutar el o los programas C#, Boo o Javascript, dentro de la escena actual, a la par de los *shaders* presentes. Cada escena en Unity se puede interpretar como "un nivel" de un videojuego.

La simulación de la escena actual, además de ejecutar los programas que se le indiquen al Motor, trabaja con diferentes módulos, que se indican a continuación:

*Motor de Física (Physx):

Unity utiliza Physx de la compañía Nvidia, el cual es básicamente un conjunto de programas que se encargan de proporcionar al motor de las herramientas necesarias para simular efectos de colisión, fricción, gravedad y aceleración en los objetos 3D que se encuentren en una escena dada. Physx es un módulo muy particular, puesto que para un programa que haga uso de él, las comprobaciones pueden realizarse vía CPU (si la computadora cuenta con aceleradora de marca AMD) o bien vía GPU en caso de que la computadora cuente con aceleradora gráfica de marca Nvidia.

*Motor de *renderización* (*Rendering Engine*):

Se encarga de realizar todos los cálculos necesarios (iluminación, materiales, sombras, efectos visuales, etc) para mostrar los gráficos de la escena actual en un dispositivo de despliegue, hace uso tanto del CPU como de la aceleradora gráfica, y de una API gráfica dependiente de la plataforma en que se vaya a liberar la aplicación final (Direct 11, Open GL, Open GL ES, etc).

*Motor de sonido:

Se encarga de realizar todos los cálculos necesarios (movimiento de los objetos emisores de sonido, de los oyentes, efectos de eco, etc) para una simulación de sonidos dentro de la escena actual. Permite importar archivos de audio en diferentes formatos para ser utilizados dentro del proyecto con diferentes opciones de calidad.

*Módulo de la UI:

Realiza todas las comprobaciones necesarias para que la interfaz de usuario del programa gráfico reaccione acorde a las entradas dadas por el usuario a través del teclado y mouse principalmente. También es capaz de reconocer entradas de dispositivos de entrada más avanzados que podemos encontrar en tabletas y teléfonos celulares, tales como los paneles táctiles multi toque, acelerómetros, etc.

Unity ejecuta también diferentes procesos en paralelo que permiten el funcionamiento del Motor, sin embargo estos no son controlables por el programador a diferencia de los módulos antes mencionados.

2) Compilador C#, intérprete de JavaScript o Boo:

Un compilador de C#, o intérprete de JavaScript y Boo que se encarga de generar los ejecutables de los programas contenidos en archivos de texto, ya sea MonoDevelop (incluido por defecto en el instalador de Unity) o Visual Studio (solo si se instaló previamente).

A diferencia del desarrollo de programas gráficos que se realizaría utilizando puramente Visual Studio, donde se utiliza una mezcla de C++ o C#, bibliotecas de diferentes herramientas útiles y llamadas a una API gráfica (Direct 3D u Open GL), en Unity todo esto se encuentra integrado al programa principal, no por ello reduciendo la eficiencia o eficacia del programa final de salida.

3DS Max 2016:

Desarrollador: Autodesk Inc

Este programa nos permite realizar modelado 3D, es decir la creación de archivos de modelos 3D utilizando una interfaz que refleja cómo se verá el modelo una vez que esté siendo *renderizado* en Unity o cualquier otro motor gráfico. Ya que en esencia, un archivo de modelos 3D no es más que una lista de vértices que componen los polígonos del modelo 3D, con indicaciones de diferentes parámetros que pueden aplicarse al modelo cuando se está *renderizando*, tales como color, material, iluminación, mapas de normales, etc. También permite realizar *renders* de calidad profesional en *hardware* adecuado.

Existen diferentes formas de crear los modelos, sin embargo, en este caso y ya que se trata de modelos con formas regulares, se diseñaron a partir de las primitivas 3D que ya provee 3DS Max y a partir de estas primitivas, se realizan las transformaciones necesarias para llegar a la forma objetivo. Finalmente se aplica un material difuso sencillo a los polígonos que correspondan y se configuran las coordenadas de texturizado adecuadas en cada caso.

Para desarrollar los modelos 3D se decidió utilizar el programa 3DS Max con una licencia de estudiante. Se decidió utilizar este programa puesto que cuenta con una interfaz considerablemente intuitiva y sencilla de aprender, comparada a las alternativas libres tales como Blender, para las cuales toma más tiempo habituarse a la interfaz y al funcionamiento del programa.

GIMP 2.8:

Desarrollador: The GIMP Team

Este programa es un editor de imágenes gratuito y libre. Se utilizó principalmente para realizar las adecuaciones necesarias a las diferentes imágenes para ser utilizadas como texturas para los modelos 3D. Principalmente se hicieron recortes y modificaciones de calidad a las imágenes, y fueron convertidas al formato PNG para su uso en Unity 3D. Se eligió este programa debido a que se realizaron dos tareas de edición de imágenes

muy simples: recortes y conversiones, por lo que no hubiera tenido sentido adquirir algunas alternativas no gratuitas tales como la paquetería Photoshop.

Microsoft Word 2013:

Desarrollador: Microsoft Corporation

Este programa se utilizó únicamente para realizar la parte escrita de la tesis, incluyendo los diagramas de bloques del funcionamiento del sistema.

MSI Afterburner:

Desarrollador: Micro-Star INT'L CO., LTD

Este programa gratuito permite realizar dos funciones: Monitoreo del estado de la CPU y de la aceleradora gráfica. Y también *overclock* sobre cualquiera de las dos componentes. Se utilizó únicamente para medir el framerate en las pruebas de rendimiento de los programas. Se utilizó este programa puesto que se consideró que era el más intuitivo de entre otras alternativas de compañías como EVGA o Asus.

6.2 *Hardware* utilizado

Computadora:

Se trabajó sobre dos computadoras para el desarrollo de los programas con las siguientes características

*Computadora 1:

-CPU Intel Core i7 6700 3.4 Ghz

-Aceleradora gráfica NVIDIA GTX 970

-16 GB RAM DDR4 2133 Mhz

-Monitor Full HD

-Almacenamiento 480 GB SSD+ 3 TB HDD

*Computadora 2:

-CPU Intel Core i7 4790 3.6 Ghz

-Aceleradora gráfica AMD R9 270

-16 GB RAM DDR3 1600 Mhz

-Monitor Full HD

-Almacenamiento 1TB HDD

Dispositivos de entrada/salida:

*Dispositivo Kinect One

*Pantalla táctil marca PQLabs

6.3 Modelo de desarrollo de *software* utilizado: *Extreme programming*

La programación extrema, XP por sus siglas en inglés, es una metodología ágil para desarrollar *software* (Fowler, 2000). **[13]**

En diferentes proyectos de distintas empresas alrededor del mundo, ha demostrado ser muy efectivo. **[9]**

Esto se debe a que se enfoca en obtener la satisfacción del cliente. En lugar de entregar todo lo que se pueda en una fecha futura, se entrega el *software* que se necesita conforme vaya siendo solicitado. **[9]**

Esta metodología empodera a los desarrolladores a responder confidentemente ante los requerimientos cambiantes del cliente incluso si estos cambios se presentan de manera tardía en el ciclo de vida del proyecto. **[9]**

Extreme programming enfatiza el trabajo en equipo. Tanto administradores de proyecto, como clientes y desarrolladores se consideran como iguales dentro del equipo colaborativo. **[9]**

De esta forma, el equipo se “auto organiza” en torno al problema para resolverlo lo más eficientemente posible, permitiendo a los equipos de trabajo volverse altamente productivos. **[9]**

En contraste con las metodologías tradicionales, burocráticas y poco ágiles, la XP tiene las siguientes características (Beck, 1999): **[13]**

1. Está orientada a quién produce y usa el *software*.

2. Reduce el costo del cambio en todas las etapas del ciclo de vida del sistema.

3. Combina las que han demostrado ser las mejores prácticas para desarrollar *software*, y las lleva al extremo.

Cuando se aplican los principios de la XP, el costo del cambio, idealmente, se mantiene constante. La XP puede describirse someramente por los siguientes puntos: **[13]**

1. Empieza en pequeño y añade funcionalidad con retroalimentación continua.

2. El manejo del cambio se convierte en parte sustantiva del proceso.

3. El costo del cambio no depende de la fase o etapa.

4. No introduce funcionalidades antes de que sean necesarias.

5. El cliente o el usuario se convierte en miembro del mismo equipo.

La XP prescribe un conjunto de valores y prácticas, que permite a los desarrolladores dedicarse a lo que hacen mejor: programar. La XP elimina la necesidad de dedicar tiempo a labores tediosas y burocráticas, prescritas por los procesos no ágiles, tales como: exhaustivos documentos de proyecto, diagramas de Gantt, enormes volúmenes de listas de requerimientos, juntas de revisión interminables, etcétera. **[13]**

Esta metodología gira en torno a cinco valores esenciales: **[9]**

1) Simplicidad: Se hará lo que se solicite y sea necesario, no más.

2) Comunicación: Todos son parte del equipo y habrá comunicación frente a frente diariamente.

3) Retroalimentación: Se tomará cada solicitud de una iteración seriamente entregando *software* funcional. Se demostrará el *software* tempranamente y a menudo, luego se escuchará con cuidado y se realizarán los cambios que se requieran. Se hablará sobre el proyecto y se adaptarán los procesos de acuerdo al mismo, no de forma inversa.

4) Respeto: Todos los miembros del equipo reciben y brindan el respeto que merecen como un miembro valioso del mismo. Todos aportan un valor incluso si es simple entusiasmo. Los desarrolladores respetan la experticia del cliente y viceversa. La administración respeta el derecho de aceptar la responsabilidad y de recibir autoridad de nuestro propio trabajo.

5) Coraje: Siempre se hablará con la verdad en cuanto a progresos y estimados. No se documentarán excusas contra fallas porque siempre se planea tener éxito. No

se temerá a nada porque nadie trabaja solo. Nos adaptaremos a los cambios cuando estos lleguen.

A pesar de lo que se pudiera pensar, en esta metodología sí se requiere que exista una comunicación constante entre clientes y desarrolladores.

Tal como se verá más adelante en las reglas del *Extreme Programming*, el programador deberá mantener un diseño simple y limpio en todo lo que haga.

Recibirá retroalimentación a partir de pruebas del *software* que desarrolla desde el primer día que comienza el proyecto. **[9]**

Además, el programador deberá entregar el sistema al cliente tan pronto como sea posible (sin ir en contra de los valores y reglas de *Extreme programming*) e implementará los cambios solicitados. **[9]**

Cada pequeño éxito durante el desarrollo del proyecto, merecerá respeto para el miembro o los miembros del equipo que sean responsables del mismo. **[9]**

De esta forma, se propone que con base a estos preceptos, los programadores que utilicen esta metodología sean capaces de responder eficientemente ante requerimientos y tecnología que cambian continuamente. **[9]**

Tal como se mencionó, *Extreme programming* dicta reglas simples basadas en valores y principios. Estas reglas no son partes aisladas sin sentido, puesto que para que sean efectivas deberán seguirse en tanto sea posible y adecuado.

<u>Las reglas</u>

1) Planeación: **[9]**
   -Se deberán escribir historias de usuario.
   -La planeación de cada entregable se utiliza para crear el calendario de entregables.
   -Los entregables son pequeños pero se realizan de manera frecuente.
   -El proyecto se divide en iteraciones
   -La planeación de iteración es lo que marca el principio de una iteración
2) Administración: **[9]**
   -El equipo de trabajo deberá tener un área de trabajo abierta, es decir, debe de estar de tal forma que el trabajo en equipo sea realmente posible y no donde cada programador está a varios metros del resto del equipo.
   -Se debe fijar un ritmo sostenible, de tal forma que cada iteración de verdad arroje *software* funcional y no programas llenos de *bugs* y errores.

-Se deberá de realizar una reunión al inicio de cada día
-La velocidad del proyecto se deberá medir
-Rotar o alternar a los programadores, de tal forma que todos sean capaces de trabajar en el código de una misma parte del proyecto, evitando el caso más común, donde una sola persona lo sabe todo de dicha parte del proyecto y nadie más
-Debemos recordar que las reglas del *Extreme programming* son una serie de herramientas que nos guían, pero que si es necesario en ciertos casos, será mejor cambiar en parte la regla para adaptarla mejor a las necesidades del proyecto y/o los programadores.

3) Diseño: **[9]**
-Ante cualquier otra cosa, la simplicidad siempre tiene prioridad
-Elegir una metáfora de sistema, es decir, una metáfora para un diseño simple con ciertas cualidades, de las cuales la más importante es que debe poder ser explicada a personas que recién se integran al proyecto sin requerir grandes cantidades de documentación para tal fin. Así, las personas que se integren al proyecto serán capaces de contribuir rápidamente. También se considera de gran importancia, la elección de nombres de clases y métodos de manera consistente. Esto por dos razones: reutilización de código y trabajo colectivo que se explican más adelante.
-Refactorizar código siempre que sea posible, no significa copiar y pegar métodos que “aparentemente funcionan”, sino que se refiere a que la redundancia y funcionalidades sin utilidad deberán ser removidas de código ya escrito, rejuveneciendo los diseños obsoletos de tal forma que sin volver a escribir todo el código, se adapte el código anterior a los requerimientos particulares del proyecto actual. Siempre manteniendo la simplicidad en que tanto se insiste.

4) Codificación: **[9]**
-El cliente debe estar disponible siempre que sea posible
-El código debe ser escrito para satisfacer estándares
-Se codifica la prueba unitaria primero
-Programación en parejas, literalmente significa que dos personas trabajan en una misma computadora para escribir código
-Solo una pareja integra código en un momento dado
-Se integra código frecuentemente
-Siempre se fija una computadora para la integración del código
-Trabajo colectivo, implica que cualquier programador puede modificar, mejorar, o corregir cualquier parte del código del proyecto

5) Pruebas: **[9]**
   -Todo el código del proyecto deberá tener pruebas unitarias
   -Todo el código del proyecto deberá pasar todas las pruebas unitarias antes de poder ser entregado
   -Cuando se encuentre un *bug*, deberán crearse pruebas
   -Las pruebas de aceptación se realizarán frecuentemente y la puntuación obtenida se hará pública

Las reglas de *Extreme programming* fijan expectativas entre los miembros del equipo pero no son el objetivo final por sí mismas. **[9]**

Se pretende entonces que estas reglas definan un ambiente donde se promueva la colaboración del equipo y el empoderamiento del mismo, los cuales son los objetivos principales. Objetivos que deberán seguir estando vigentes aún en el punto en que el equipo de trabajo alcance un nivel de productividad muy alto. **[9]**

Debemos recordar que no existe una metodología mejor que otra, sino que cada metodología es más adecuada que otra para cada proyecto en particular donde se vaya a utilizar. Dependerá tanto del proyecto en sí mismo, como del equipo de trabajo que esté desarrollando el proyecto.

Se utilizó esta metodología por tres razones principales:

-Cambios en los requerimientos impredecibles: durante todo el tiempo que se desarrolló el sistema, los requerimientos iniciales estuvieron cambiando continuamente, por lo que una metodología clásica como cascada que necesita un listado de requerimientos bien definidos sencillamente no hubiera resultado adecuada.

-Equipo de trabajo reducido: únicamente el autor de la presente tesis trabajó directamente en el desarrollo del código fuente necesario para el sistema, motivo por el cual metodologías que tienden a necesitar muchas personas solo para el proceso de administración del equipo de trabajo sencillamente hubieran sido imposibles de aplicar.

-Adicionalmente, durante el tiempo que se estuvo desarrollando el sistema, por cuestiones de tiempo y con el fin de agilizar el desarrollo, lo más simple fue basarse en un sistema donde se desarrollaran programas entregables funcionales de acuerdo a los requerimientos actuales que se hubieran planteado en cada reunión mensual en el instituto.

6.4 Proceso de desarrollo de los programas

En cuanto a los sistemas que se comunican con el *software*, el *Kinect* y la pantalla táctil se hizo lo siguiente:

-Pantalla táctil:

En primer lugar, es importante mencionar las características esenciales de la pantalla táctil puesto que esto explicará el porqué del proceder para desarrollar la solución al problema de la integración de la pantalla con el sistema.

La pantalla táctil, utiliza tecnología de sensor-emisor infrarrojo para la detección de los toques que el usuario efectúe sobre la superficie de la pantalla. Esto causa un ligero desfasamiento en la posición real de los dedos del usuario con respecto a la posición que se recibe dentro del programa, además de un ligero retardo para detectar movimientos rápidos. Finalmente, el sensor infrarrojo está limitado de tal forma que solo es posible detectar dos dedos o dos toques independientes en la superficie de la pantalla, de modo que cualquier toque adicional causará que se estén interpretando diferentes dedos del usuario o incluso objetos sobre la superficie como los “dos toques actuales”. Adicionalmente, es importante mencionar que se comunica con la computadora a través de un puerto USB 2.0, para ello requiere un driver provisto por el fabricante de la pantalla en su página oficial que permite realizar ciertas configuraciones y calibraciones.

En principio y tal como se propuso, se utilizaron los métodos nativos de C# en Unity para manejar las entradas táctiles, lo cual daba resultados aceptables. El problema mayor es que a pesar de que la pantalla táctil cuenta con detección de dos puntos de toque independientes, por alguna razón los métodos nativos de Unity solo estaban detectando el primer toque que se efectuaba sobre la pantalla, el segundo toque se ignoraba y el colocar simultáneamente los dos dedos sobre la pantalla causaba un error que evitaba la detección correcta de las posiciones de los dos toques. Por esta razón se tuvo que recurrir a soluciones alternativas que se pueden encontrar en la página del fabricante. La que mejor funcionó, fue un *plugin* basado en TUIO denominado “unity3d-tuio” de la empresa *Mindstorm* bajo la licencia LGNU. Estableciendo comunicación entre los programas propios y el *plugin* fue posible detectar efectivamente los dos toques sobre la pantalla sin ningún problema, así como obtener un conteo en tiempo real de la cantidad de toques en cierto momento que se necesitará. De esta manera, el programa funcionó considerablemente mejor tanto para uno como los dos toques sobre la pantalla.

-*Kinect One*:

El *Kinect One* es un dispositivo diseñado y fabricado por Microsoft, que cuenta con diferentes sensores que le permiten detectar los movimientos de una o más personas con buena precisión y en tanto se mantenga una distancia adecuada entre las personas y el dispositivo. Adicionalmente cuenta con cámara RGB de alta definición, cámara de definición estándar de infrarrojo activo (similar a las cámaras de visión nocturna) y tres micrófonos de alta sensibilidad, que se utilizan principalmente para comandos por voz.

En particular, para la presente tesis se utiliza la capacidad del dispositivo para detectar los movimientos del cuerpo de una persona, con el fin de que el dispositivo cumpla la función de entrada para el programa. El SDK y el paquete provisto por Microsoft para Unity 3D cumple una función muy importante, puesto que se encarga de manejar a nivel *hardware* las señales que recibe el sensor y transformarlas al final de todo el proceso, en estructuras de datos que C# y Unity 3D pueden trabajar como un tipo de dato más. De esta forma, Microsoft provee lo necesario para poder desarrollar un programa con objetivos más específicos.

Se utilizaron las herramientas de Microsoft para manejar la parte de *hardware* y utilizar la salida final del proceso, con diferentes modificaciones al programa base que provee Microsoft para así poder detectar la posición de la mano de un paciente, recibir dicha información en el programa principal del sistema y así realizar la misma función de la pantalla táctil para mover la pelota o el paño dependiendo del ejercicio que se tratara.

A diferencia de la pantalla táctil, el sensor *Kinect* no tiene puntos de toque fijos, sino que detecta todo el cuerpo del usuario, lo divide en un “esqueleto” que consta de las distintas articulaciones del cuerpo humano y almacena en diferentes variables la posición que cada articulación tiene cada vez que se llama la función que hace el refrescado. Las articulaciones detectadas son:

-Cabeza (se detecta toda la cabeza como un solo punto en el espacio)

-Rostro (en el modo usual, solo se detecta la boca y la nariz, aunque es posible que el dispositivo haga un reconocimiento detallado basado en puntos distintos del rostro que incluso permite reconocer gestos faciales)

-Cuello

-Hombros (se detectan ambos de manera simultánea)

-Codos (se detectan ambos de manera simultánea)

-Muñecas

-Manos

-Dedo pulgar (este dedo se detecta de manera independiente al resto de los dedos de la mano) de la manos derecha y de la mano izquierda

-Dedos (los dedos índice, anular y meñique se detectan como un solo punto) de la mano izquierda y derecha

En particular, se utilizan solamente las manos del usuario como punto de referencia para mover el paño o la pelota acorde a los movimientos del usuario en el mundo real. Esto se decidió así puesto que al realizar diferentes pruebas con otras articulaciones, como los dedos, las muñecas o el dedo pulgar no resultaba natural el movimiento del paño o la pelota dentro del programa, sin mencionar que ocurren más errores de oclusión cuando se utiliza una articulación pequeña como los dedos. Con error de “oclusión”, se refiere a aquellos casos en que por alguna razón una articulación se sitúa enfrente a otra respecto del punto de vista del *Kinect*, esto provoca que el dispositivo tenga problemas para detectar correctamente la posición de la articulación ocluida y por tanto el comportamiento del programa no sea óptimo en cuanto a la forma en que responde ante el usuario.

De este modo, al recibir la posición de la mano del usuario, es posible mover el paño o la pelota según se requiera.

7. Diseño del sistema

7.1 *Hardware*

En el presente desarrollo, se diseñaron dos modos en que el sistema puede utilizarse, uno es utilizando computadora y pantalla táctil, y el segundo modo es computadora, pantalla táctil y *Kinect One*.

Modo Computadora-Pantalla táctil

Para el modo “computadora y pantalla táctil” se utilizan dos dispositivos:

-Computadora:

Esta es la computadora que se encarga de ejecutar los programas gráficos, de desplegar las imágenes correspondientes en la pantalla (dispositivo de salida) y de responder de acuerdo a los datos que se reciben por el sensor de la pantalla táctil (dispositivo de entrada).

-Pantalla táctil: este dispositivo realiza dos funciones, como dispositivo de entrada y salida. Permite desplegar imágenes en una resolución máxima de 1600 x 900 píxeles a 60 Hz. Y cuenta con un sistema basado en un emisor y un sensor infrarrojo, que le permite detectar si algún objeto está haciendo contacto con la pantalla, únicamente puede detectar dos puntos de contacto simultáneamente. Envía la información de la posición y estado de los toques a través de un cable USB 2.0 hacia la computadora, interpretado normalmente por la computadora como un “segundo *mouse*”.

Modo Computadora-Pantalla-*Kinect*

Para el modo “computadora, pantalla y *Kinect*” se utilizan tres dispositivos:

-Computadora: esta computadora se encarga de ejecutar los programas gráficos, de desplegar las imágenes correspondientes en el proyector y de responder de acuerdo a los datos que se reciben por el sensor *Kinect*.

-Pantalla: este dispositivo de salida se encarga del despliegue de las imágenes del programa, y a diferencia del modo anterior, en este caso no se requiere que cuente con capacidad de detección de toques ya que es el sensor *Kinect* el que se encarga de detectar los movimientos de la mano del usuario.

-*Kinect One*: este dispositivo de entrada, cuenta con sensores y emisores infrarrojos que le permiten detectar los movimientos del cuerpo de una persona, también cuenta con cámara a color (RGB 24 bits) de alta definición y una cámara sensible al infrarrojo

cercano. En el presente desarrollo, únicamente se hace uso de la capacidad del dispositivo para detectar los movimientos de la mano del paciente, misma información que el programa recibirá en la computadora para reaccionar acorde a las acciones del paciente.

7.2 *Software*

Sistema Operativo

El presente desarrollo fue diseñado y probado únicamente para Windows 8, 8.1 y 10.

No debemos olvidar, que el *Kinect One* únicamente es compatible con Windows 8, 8.1 y 10 (de manera ordinaria).

Programas

Se desarrollaron 4 programas ejecutables en total, los cuales en realidad son dos programas en sus versiones respectivas para *Kinect* y pantalla táctil. Es decir, se tiene el programa de minigolf para pantalla táctil, el programa de minigolf para sensor *Kinect*, el programa de limpiar la ventana para pantalla táctil y el programa de limpiar ventana para sensor *Kinect*.

***Programa de minigolf:**

Consiste de un circuito de minigolf donde el paciente debe llevar la pelota hacia el hueco del circuito. Basándose en los ejercicios presentados en el libro mencionado anteriormente, contiene 3 ejercicios distintos, con dos niveles de dificultad cada uno.

Nivel de dificultad normal (con canaleta)

En el primer nivel todos los circuitos tienen una canaleta en los bordes que sirve de guía al paciente para saber por dónde llevar la pelota, adicionalmente existe una línea blanca en el suelo del circuito que indica también en qué dirección llevar la pelota. Si el paciente hace chocar la pelota con la canaleta esto es penalizado como un error (un punto), adicionalmente la pelota toma una coloración roja (Rojo puro RGB) para indicar al paciente que la pelota ha chocado. En las figuras 7.2.1, 7.2.3 y 7.2.5 se pueden observar los programas funcionando en este modo.

Nivel de dificultad mayor (sin canaleta)

En este nivel sigue existiendo una línea blanca en el suelo de los circuitos, sin embargo ahora no existe ninguna canaleta en los bordes de los circuitos, por lo cual el paciente deberá llevar la pelota justo sobre la línea blanca de guía sin salirse. El sacar la pelota de la línea blanca o bien llevarla al agujero incorrecto es penalizado como un error (un

punto), adicionalmente la pelota toma una coloración roja (Rojo puro RGB) para indicar al paciente que ha salido de la línea de guía. En las figuras 7.2.2, 7.2.4 y 7.2.6 se pueden observar los ejercicios funcionando en este modo.

Ejercicios:

1) Horizontal: El primer ejercicio tiene como objetivo practicar el movimiento de la mano del paciente de izquierda a derecha, motivo por el cual, este ejercicio consiste de pistas o canaletas dispuestas en forma horizontal con el agujero situado del lado derecho, tal como se observa en las Figura 7.2.1 y 7.2.2 De esta forma el paciente lleva la pelota del inicio de la pista o canaleta hacia el hueco de la derecha.

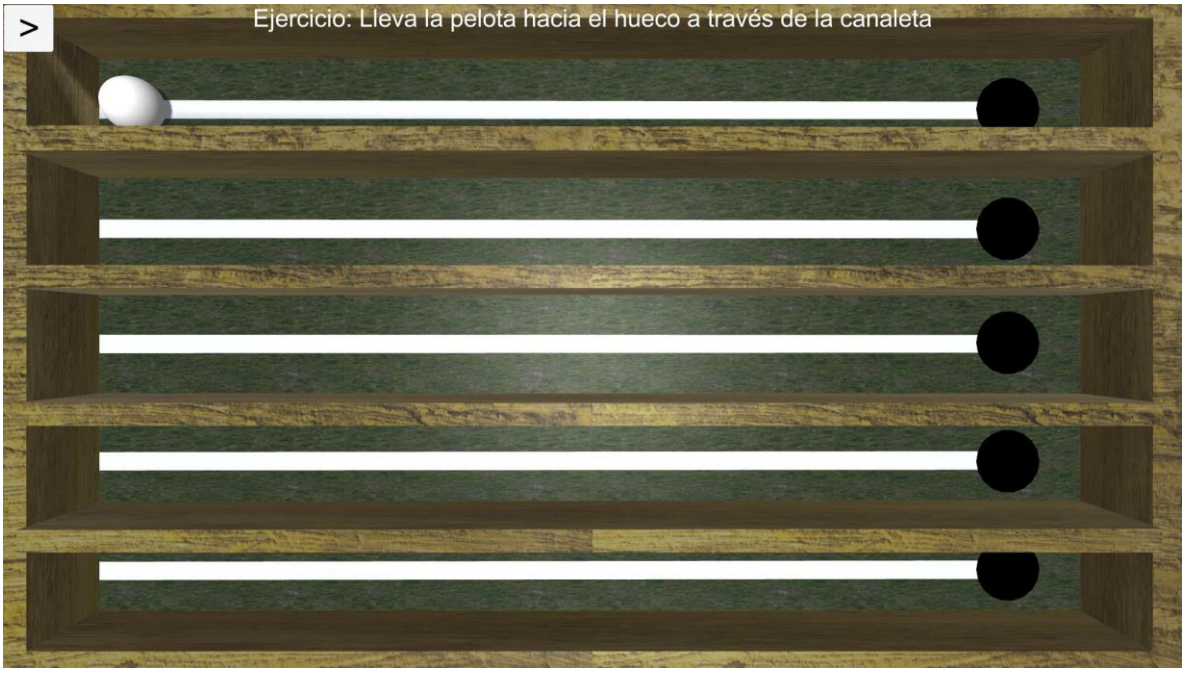


Figura 7.2.1 Programa de minigolf ejecutándose en modo horizontal y dificultad con canaleta

Figura 7.2.2 Programa de minigolf ejecutándose en modo horizontal y dificultad sin canaleta

2) Vertical: El segundo ejercicio tiene como objetivo practicar el movimiento de la mano del paciente de arriba hacia abajo, motivo por el cual, este ejercicio consiste de pistas o canaletas dispuestas en forma vertical, con el agujero situado en la parte baja de la canaleta, tal como se observa en las figuras 7.2.3 y 7.2.4. De esta forma el paciente lleva la pelota del inicio de la pista o canaleta hacia el hueco de abajo.

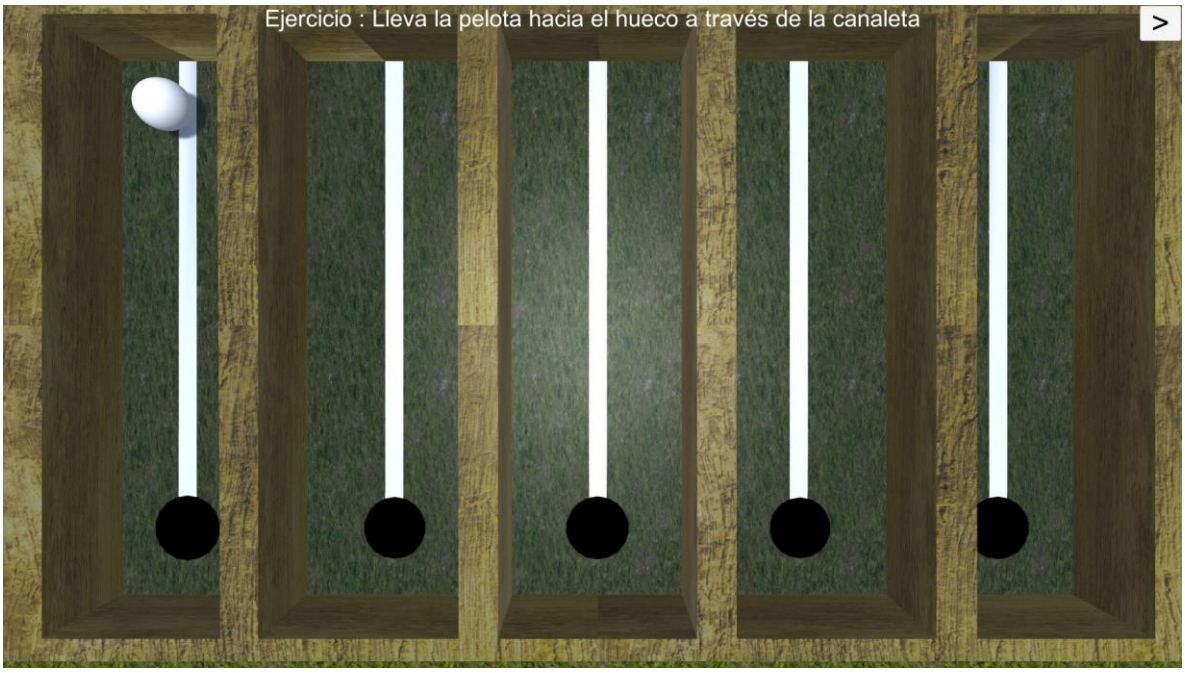


Figura 7.2.3 Programa de minigolf ejecutándose en modo vertical y dificultad con canaleta

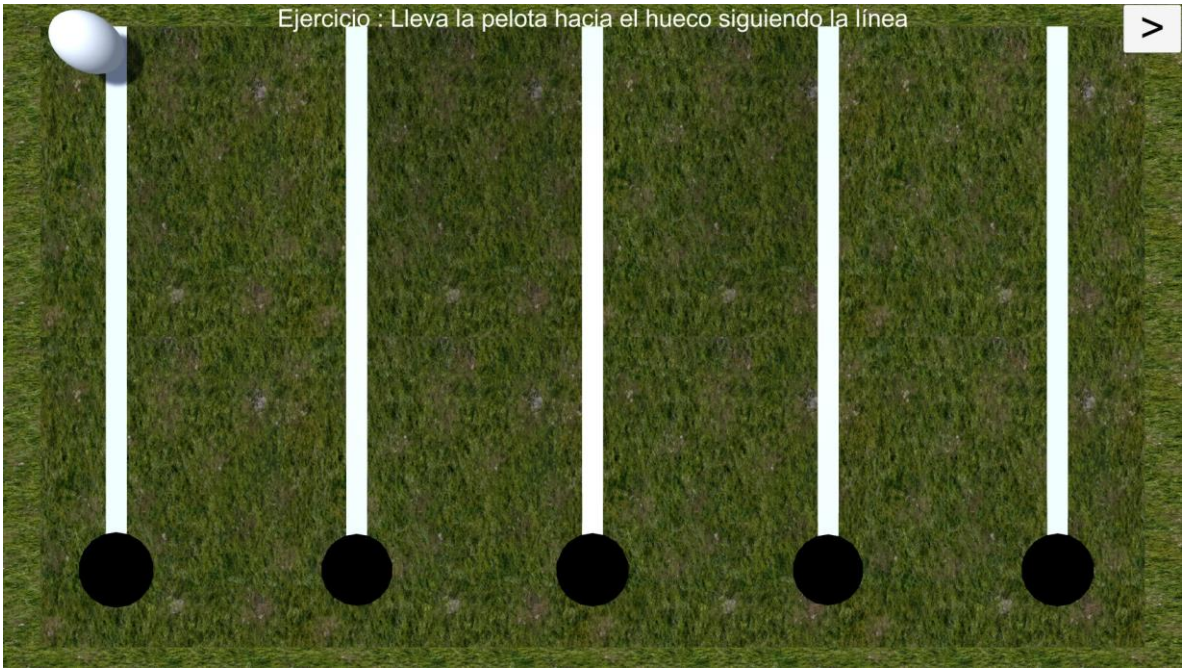


Figura 7.2.4 Programa de minigolf ejecutándose en modo vertical y dificultad sin canaleta

3) Combinado: Este ejercicio combina tanto movimientos de izquierda a derecha, arriba hacia abajo y abajo hacia arriba, mientras que el hueco se encuentra situado del lado derecho de la pista o canaleta tal como se observa en las figuras 7.2.5 y 7.2.6. Sin embargo para llevar la pelota hasta el hueco se requiere combinar diferentes movimientos.

Figura 7.2.5 Programa de minigolf ejecutándose en modo combinado y dificultad con canaleta

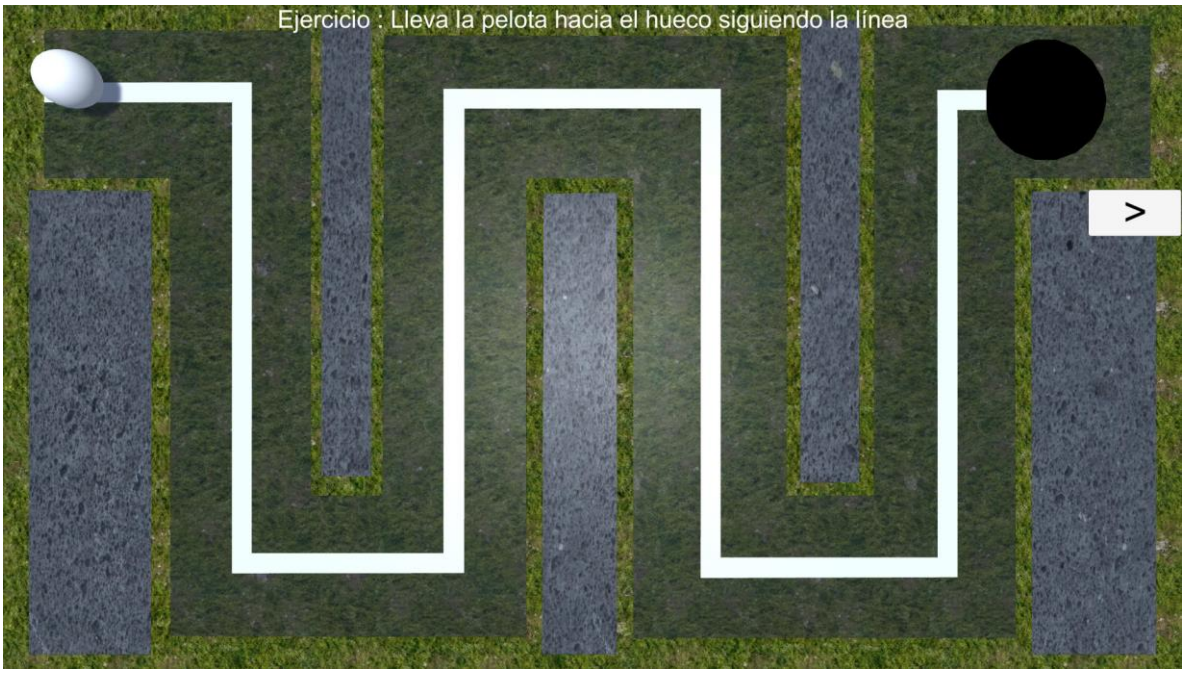


Figura 7.2.6 Programa de minigolf ejecutándose en modo combinado y dificultad sin canaleta

Repeticiones:

La idea básica de las repeticiones, es que se da un número de las que se desea que el paciente realice y se introduce este número en el programa en el menú de configuración al inicio del mismo. En el caso de los ejercicios horizontal y vertical, el programa contará tantas veces como sea necesario llevar la pelota al hueco que corresponda a la canaleta o pista actual hasta completar las repeticiones indicadas, así por ejemplo, en 5 repeticiones, el paciente deberá llevar la pelota, por 5 canaletas o pistas hasta el hueco correspondiente de cada una y hasta entonces el ejercicio terminará. En el caso del ejercicio combinado, una repetición consistirá en llevar la pelota desde el inicio de la canaleta o pista hasta el hueco del circuito, momento en el cual se contará como una repetición.

Una vez que se finaliza cualquier ejercicio de cualquier nivel de dificultad, se indica en pantalla la cantidad de errores por repetición que se cometieron. De esta forma el paciente podrá proponerse (o bien el terapeuta exigirle al paciente) reducir el número de

errores cometidos hasta llegar hasta un teórico total de cero errores para una persona sana.

***Programa de limpiar la ventana:**

El paciente debe limpiar las manchas de suciedad que se encuentran sobre la ventana mostrada en el menor tiempo posible, y siguiendo una dirección indicada desde el principio.

Ejercicios:

-Horizontal: De manera similar al programa de minigolf, se colocan las manchas de suciedad en una disposición adecuada para que el paciente las limpie mediante movimientos de izquierda a derecha tal como se observa en la figura 7.2.7 y se detecta la dirección en que el paciente limpia las manchas para asegurar que está realizando el ejercicio en forma correcta, de no hacerlo sencillamente la mancha no se limpia o no desaparece al pasar el paño encima.

Figura 7.2.7 Programa de limpiar la ventana en modo horizontal

-Vertical: se colocan las manchas en una disposición adecuada para que el paciente las limpie mediante movimientos de arriba hacia abajo tal como se observa en la figura 7.2.8. De igual forma se detecta la dirección del movimiento del paño para asegurar que esta sea correcta de lo contrario la mancha no se limpia al pasar el paño.

Figura 7.2.8 Programa de limpiar la ventana en modo vertical

-Circular: en este caso, las manchas se colocan en una especie de espiral con el fin de que el paciente se vea obligado a realizar movimientos circulares o elípticos para limpiar las manchas en un orden preestablecido tal como se observa en la figura 7.2.9, y de no seguir el orden correcto las manchas no se eliminan al pasar el paño por encima.

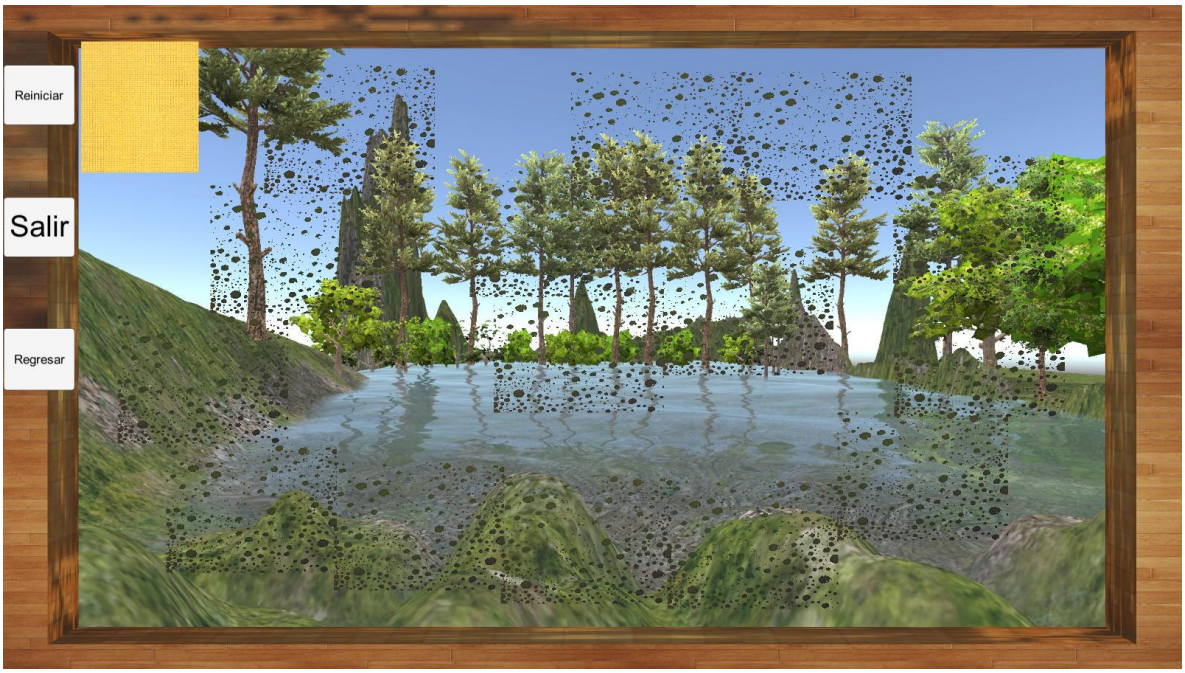


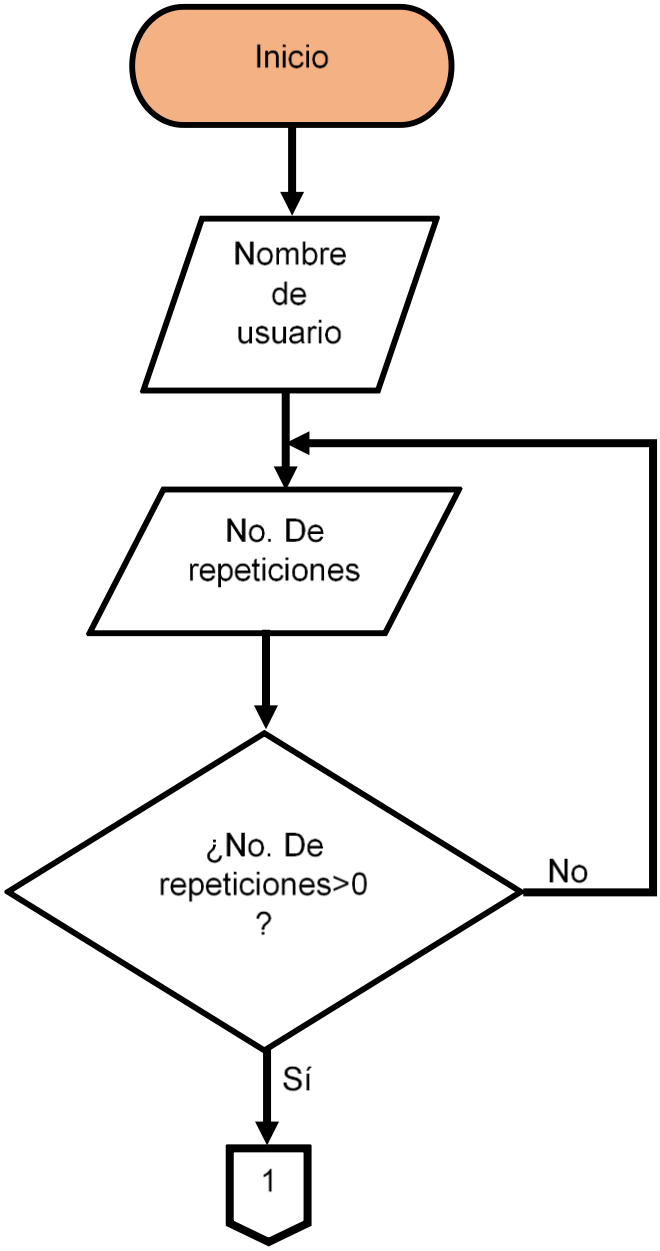
Figura 7.2.9 Programa de limpiar la ventana en modo círculos

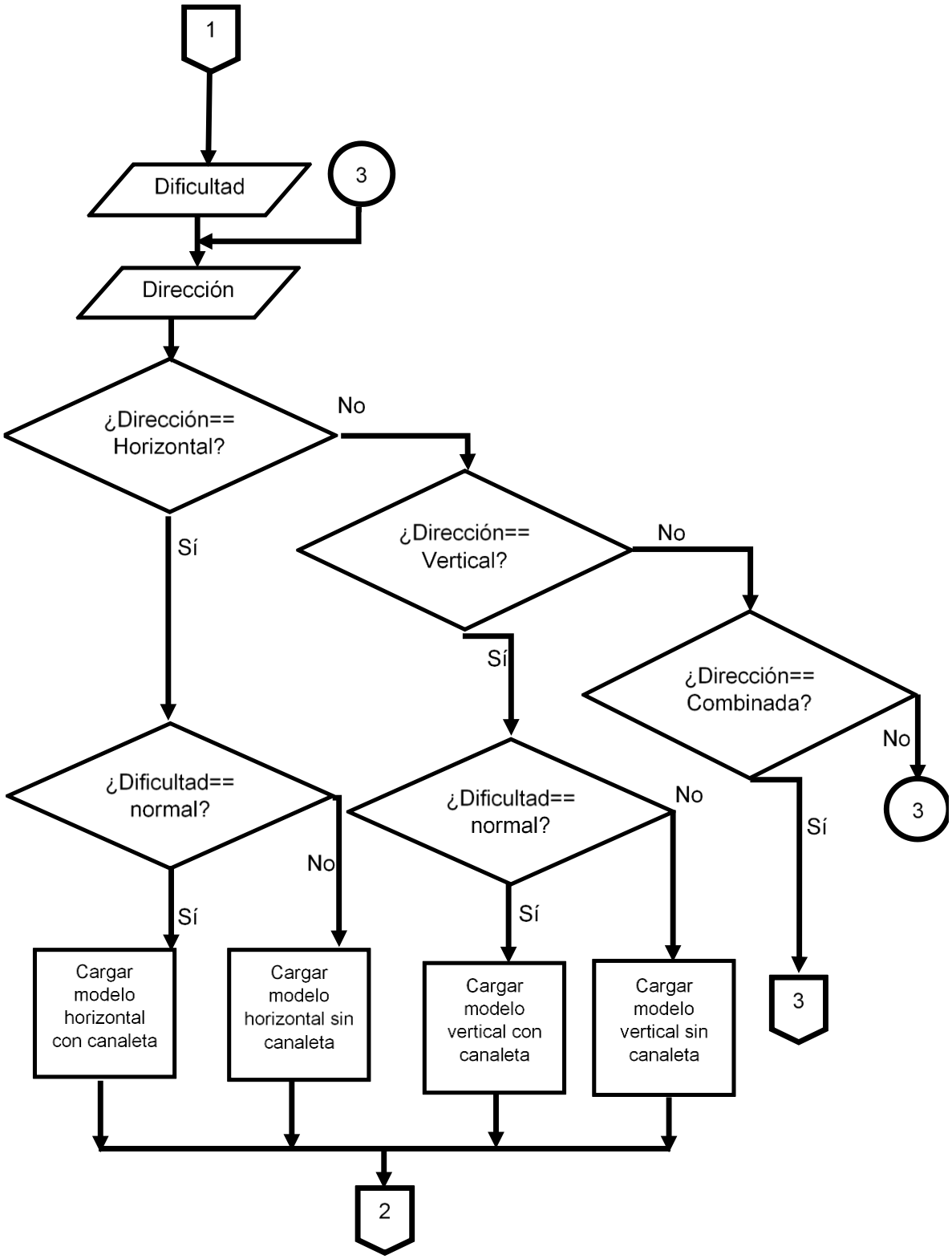
Es posible indicar cuantas manchas aparecen en la pantalla, para de esta forma ir incrementando el número de manchas que el paciente debe limpiar en el menor tiempo posible para cada dirección dada.

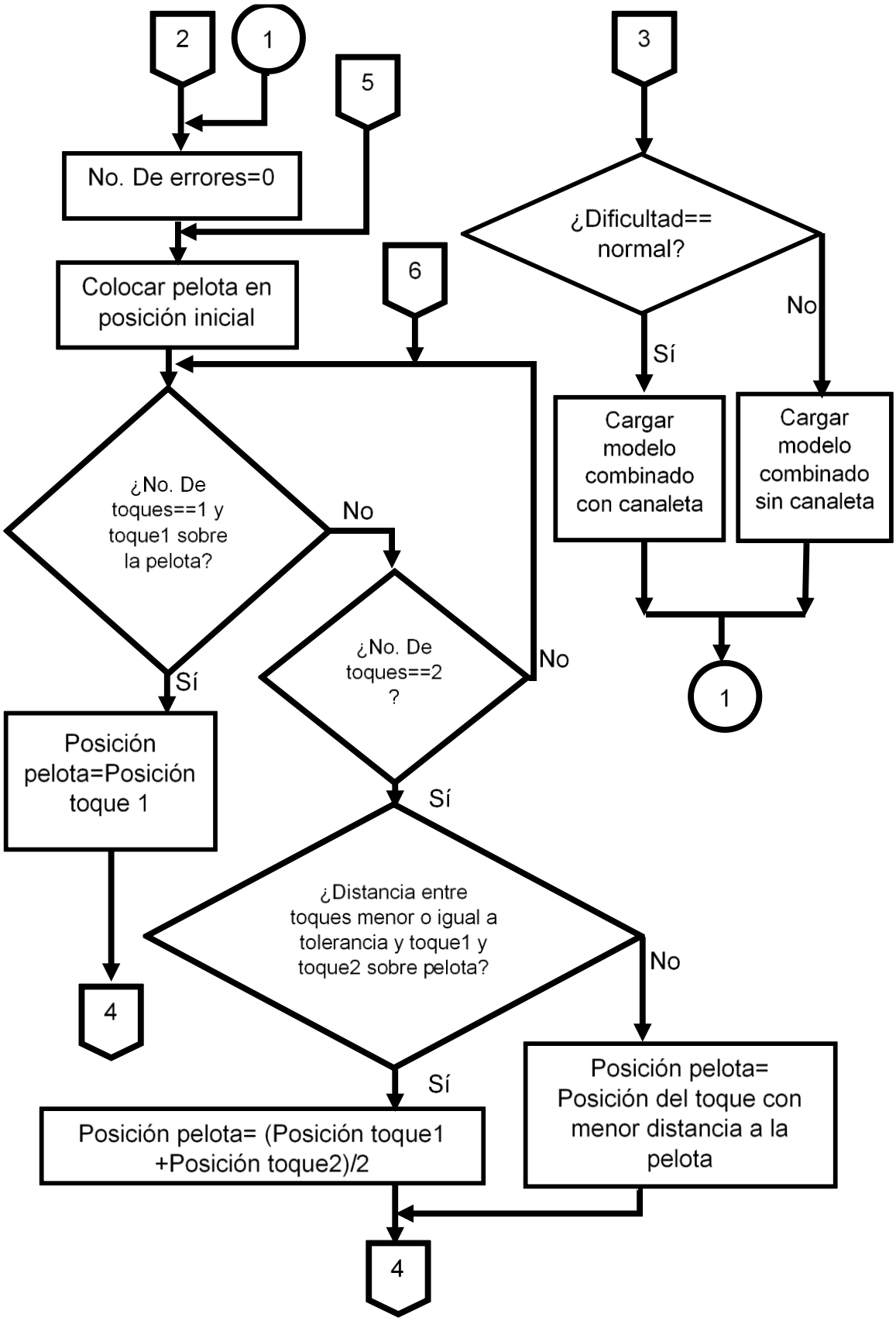
Para evaluar el desempeño del paciente se cuenta el tiempo, desde que el trapo toca una mancha hasta que se limpia la última de ellas. Dicho tiempo es presentado en forma de minutos y segundos al finalizar el ejercicio. De esta forma el terapeuta le propondrá al paciente tratar de limpiar un cierto número de manchas en un tiempo cada vez menor.

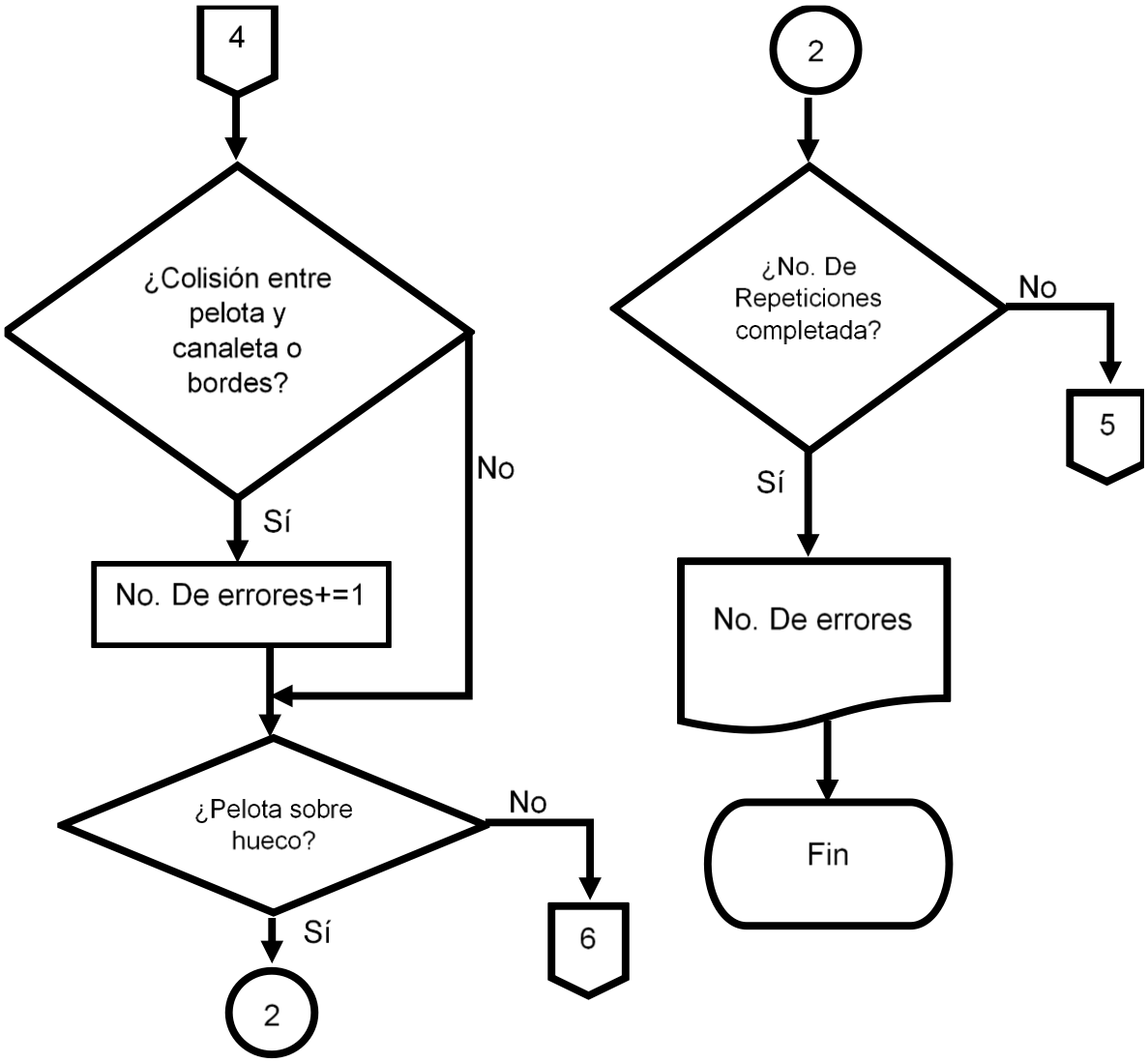
7.3 Diagramas

-Programa de minigolf, versión para pantalla táctil:



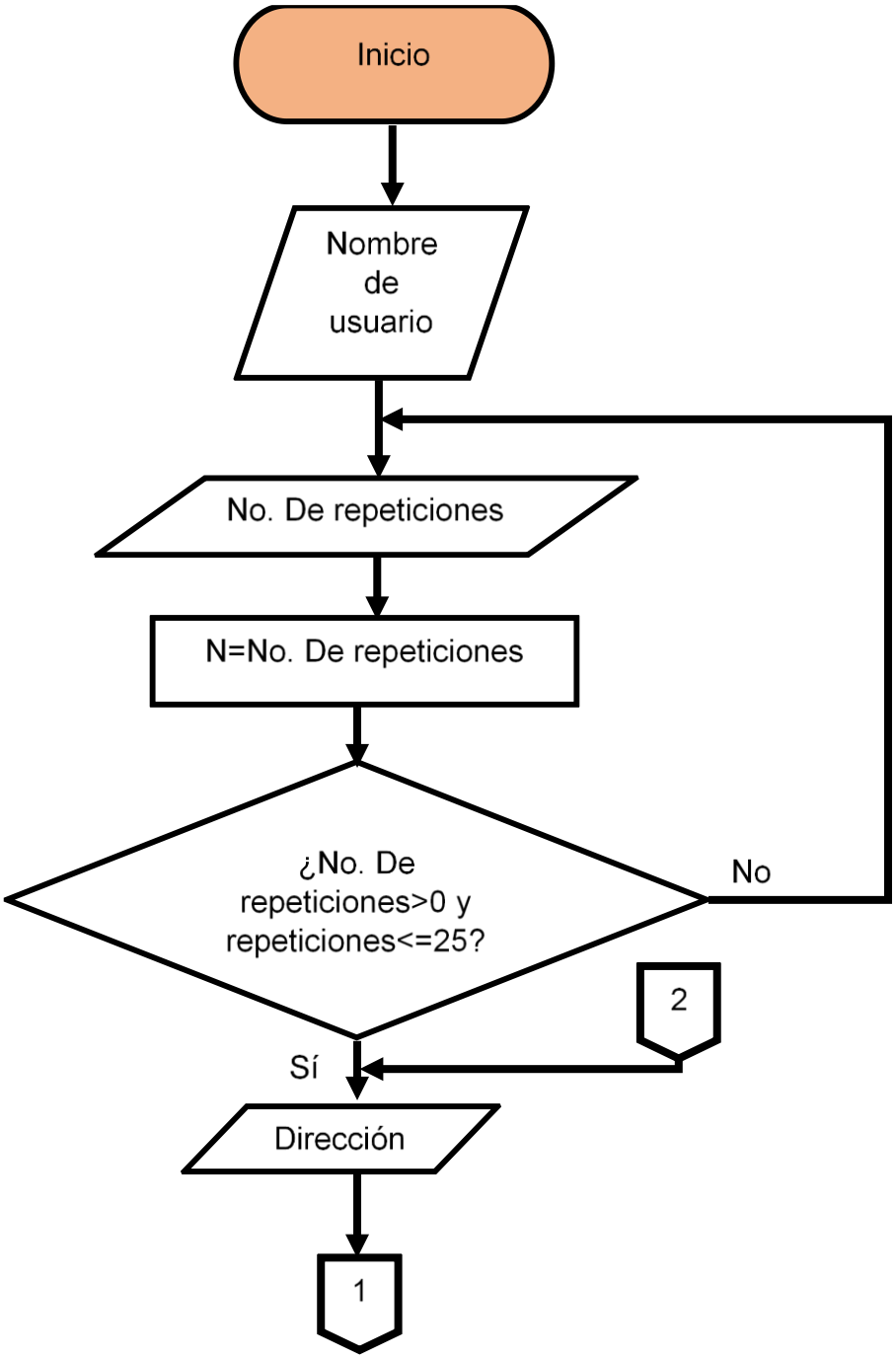

1
Dificultad
3
Dirección
¿Dirección== Horizontal?
No
Sí
¿Dirección== Vertical?
No
Sí
¿Dirección== Combinada?
No
Sí
3
¿Dificultad== normal?
Sí
No
¿Dificultad== normal?
No
Sí
Cargar modelo horizontal con canaleta
Cargar modelo horizontal sin canaleta
Cargar modelo vertical con canaleta
Cargar modelo vertical sin canaleta
3
2

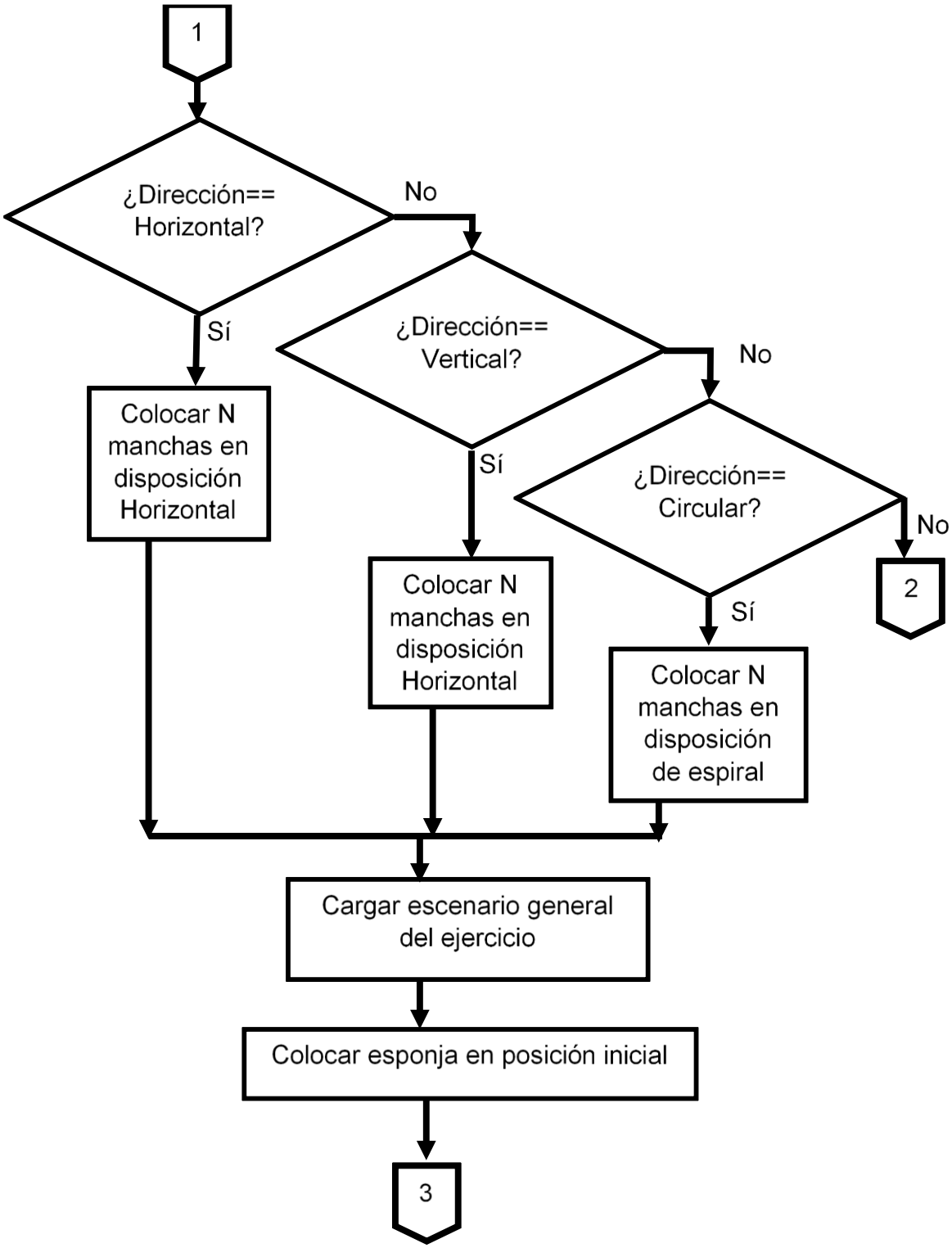

2
1
5
3
No. De errores=0
¿Dificultad== normal?
No
Sí
6
Colocar pelota en posición inicial
Cargar modelo combinado con canaleta
Cargar modelo combinado sin canaleta
¿No. De toques==1 y toque1 sobre la pelota?
No
¿No. De toques==2 ?
No
Sí
1
Posición pelota=Posición toque 1
Sí
¿Distancia entre toques menor o igual a tolerancia y toque1 y toque2 sobre pelota?
No
4
Sí
Posición pelota= Posición del toque con menor distancia a la pelota
Posición pelota= (Posición toque1 +Posición toque2)/2
4

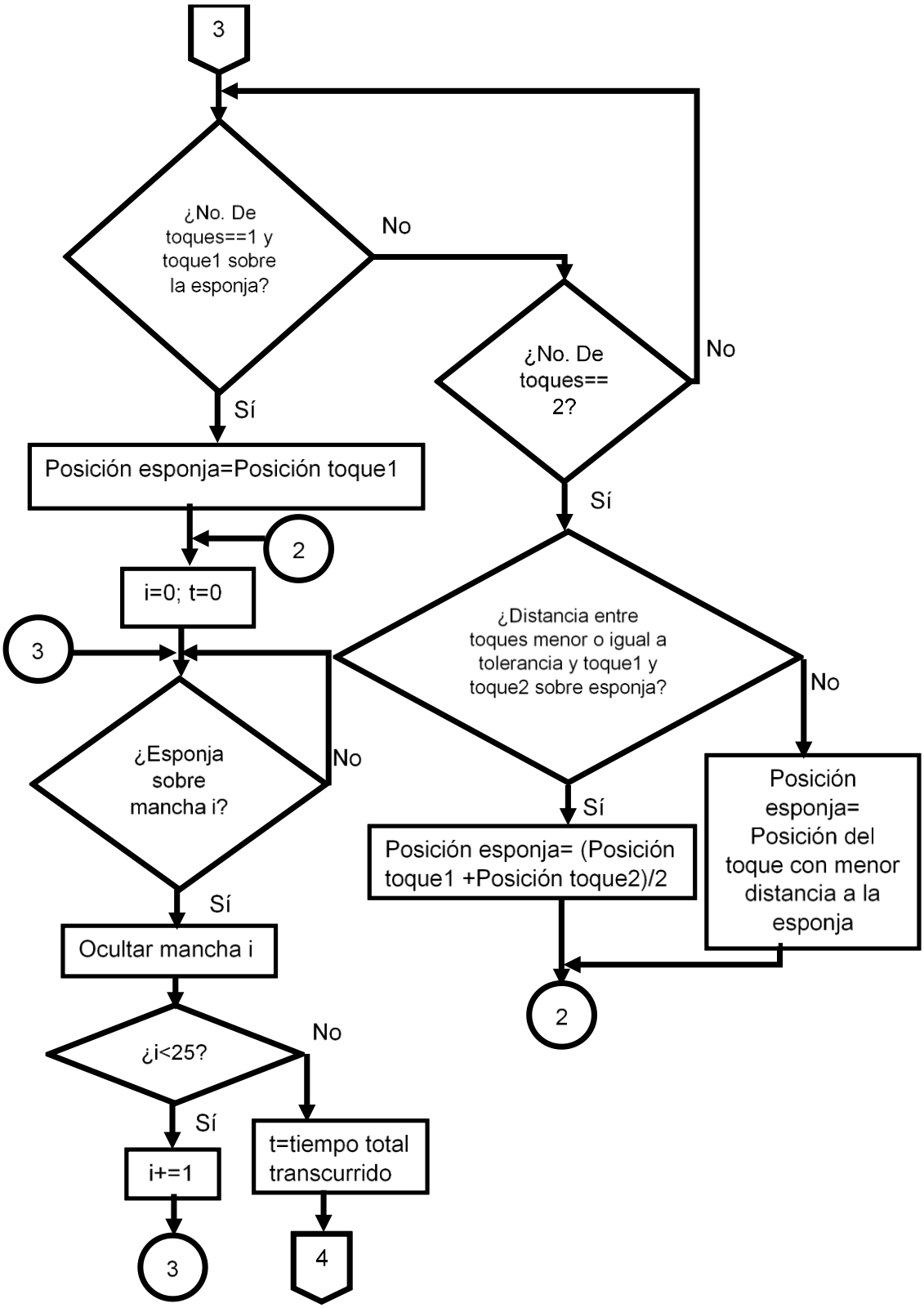


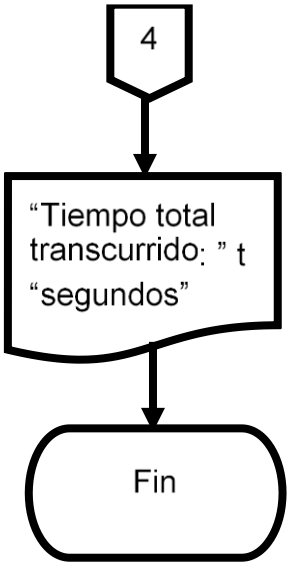
Los scripts en lenguaje C# correspondientes al programa de minigolf para pantalla táctil, se encuentran en el apéndice 1.

-Programa de limpiar la ventana, versión para pantalla táctil:



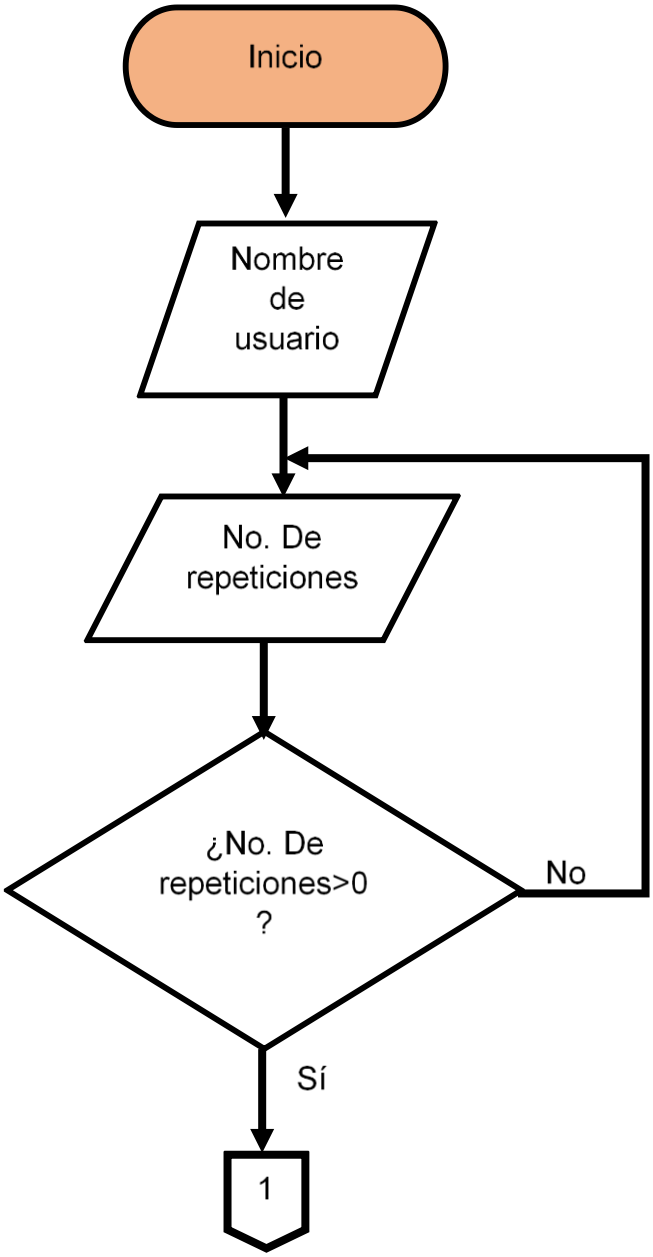

1
¿Dirección== Horizontal?
No
Sí
¿Dirección== Vertical?
No
Sí
¿Dirección== Circular?
No
Sí
Colocar N manchas en disposición Horizontal
Colocar N manchas en disposición Horizontal
Colocar N manchas en disposición de espiral
2
Cargar escenario general del ejercicio
Colocar esponja en posición inicial
3

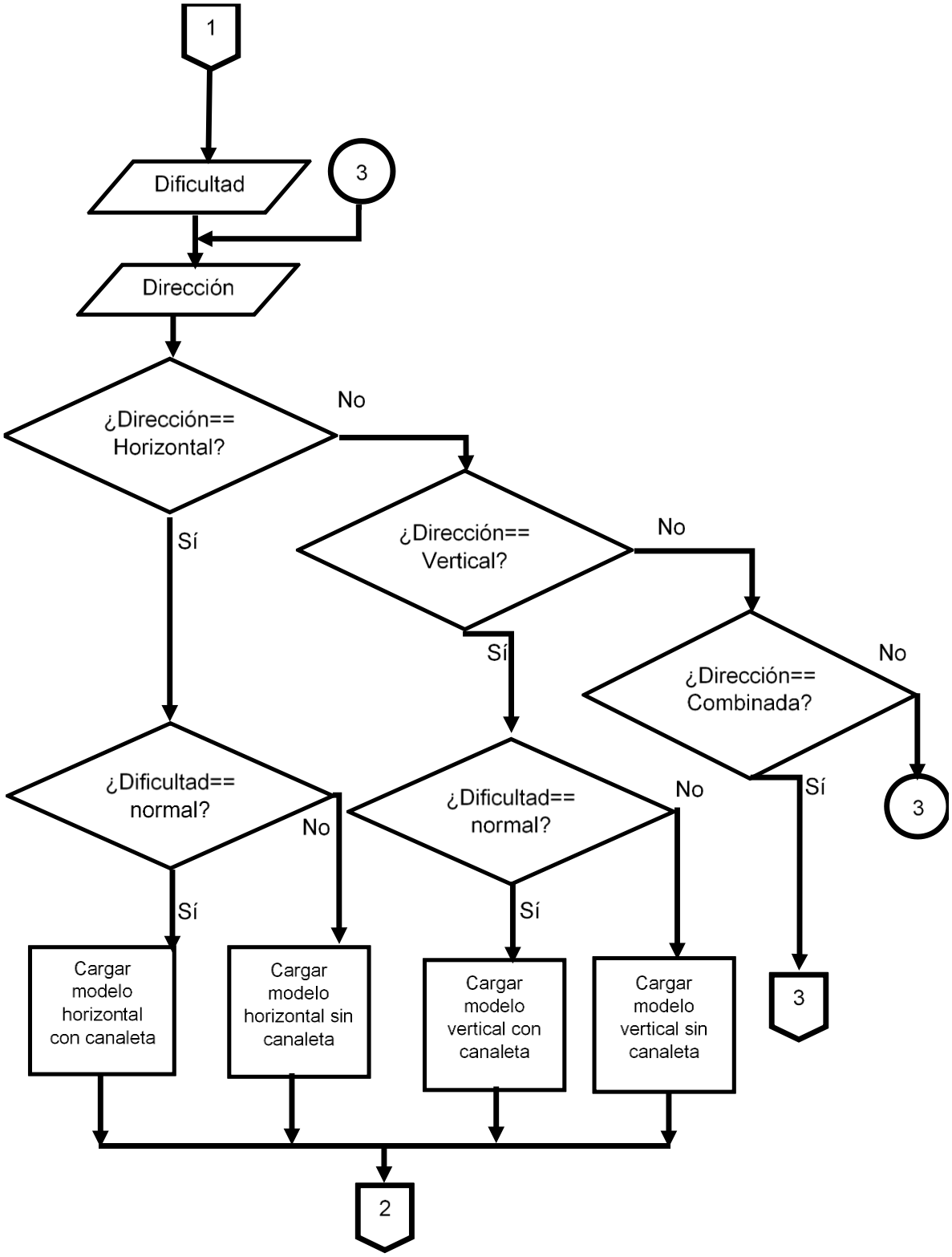

3
¿No. De toques==1 y toque1 sobre la esponja?
No
¿No. De toques== 2?
No
Sí
Posición esponja=Posición toque1
Sí
2
i=0; t=0
¿Distancia entre toques menor o igual a tolerancia y toque1 y toque2 sobre esponja?
No
3
¿Esponja sobre mancha i?
No
Posición esponja= Posición del toque con menor distancia a la esponja
Sí
Posición esponja= (Posición toque1 +Posición toque2)/2
Sí
Ocultar mancha i
2
¿i<25?
No
Sí
t=tiempo total transcurrido
i+=1
3
4

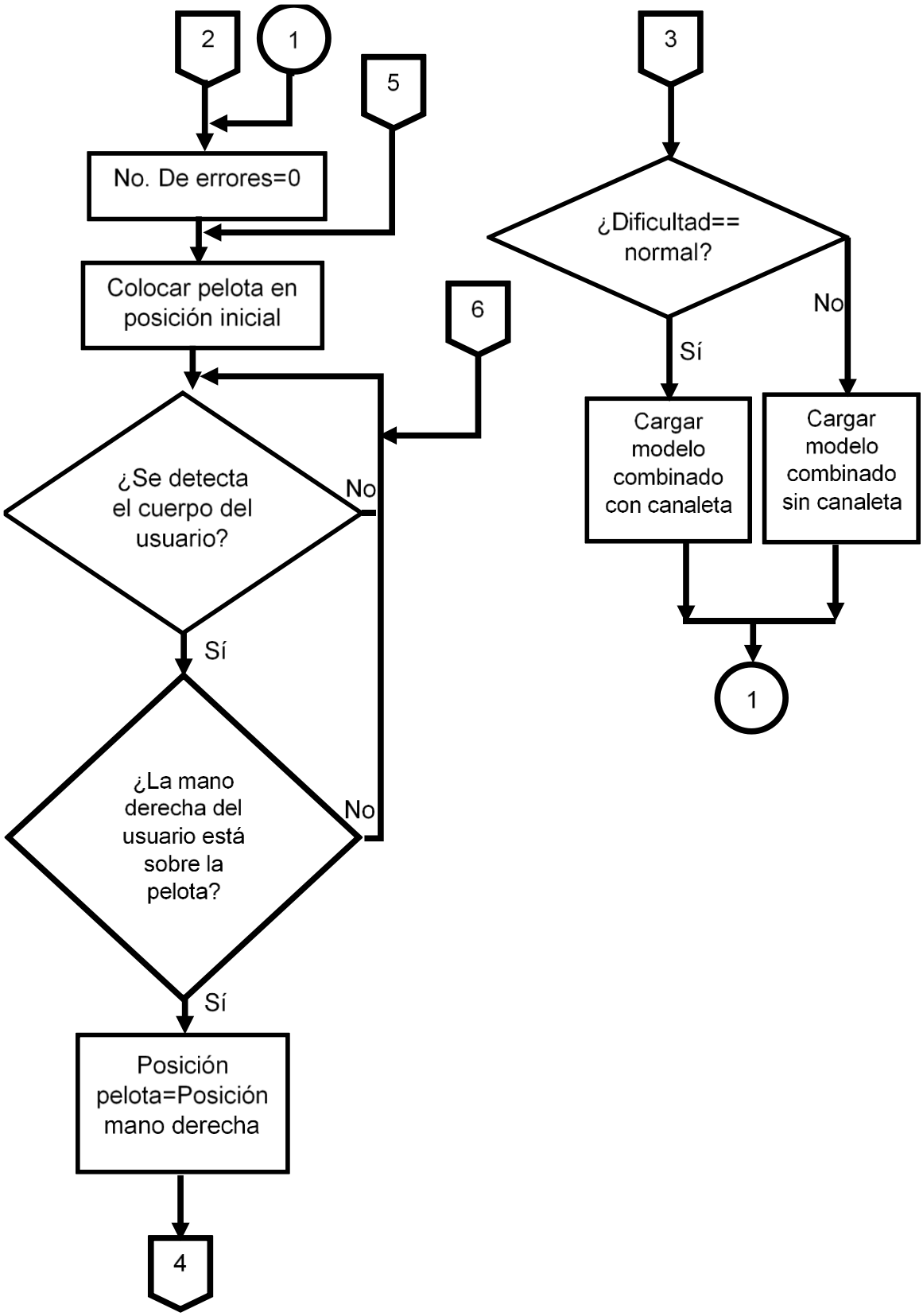


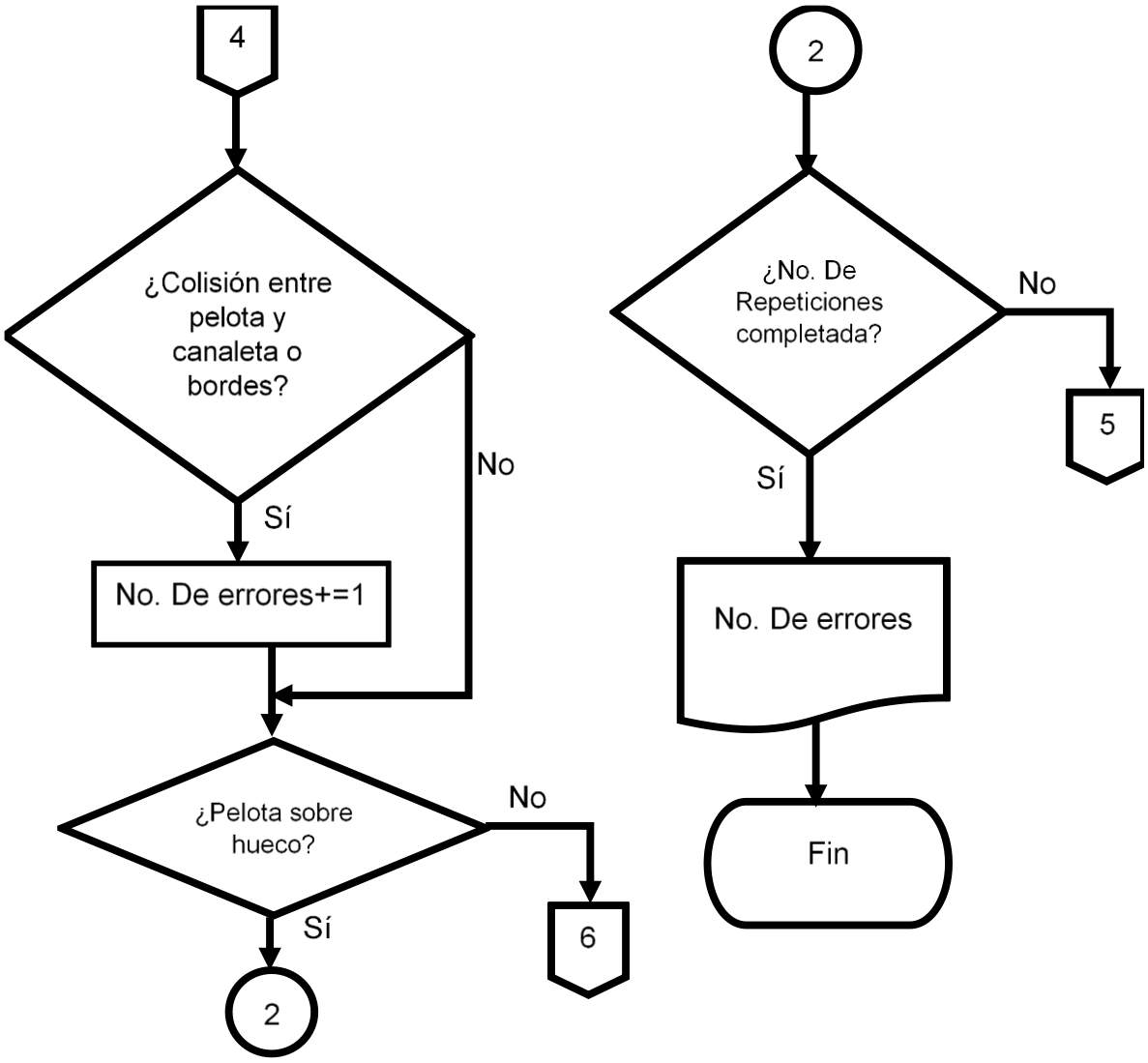
Los scripts en lenguaje C# correspondientes al programa de limpiar la ventana para pantalla táctil se encuentran en el apéndice 2.

-Programa de minigolf, versión para sensor *Kinect One*:



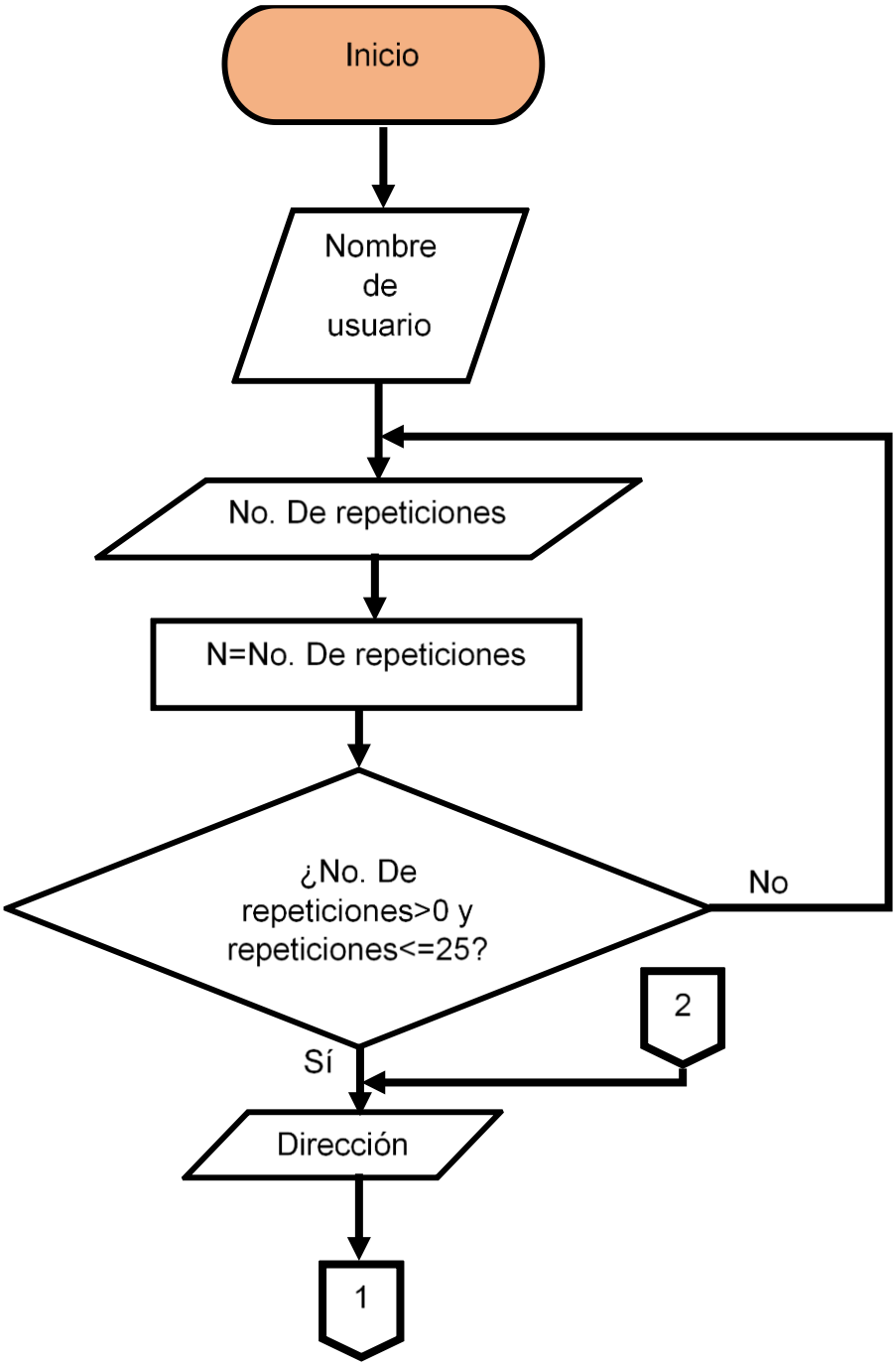

1
Dificultad
3
Dirección
¿Dirección== Horizontal?
No
Sí
¿Dirección== Vertical?
No
Sí
¿Dirección== Combinada?
No
Sí
3
¿Dificultad== normal?
No
Sí
¿Dificultad== normal?
No
Sí
Cargar modelo horizontal con canaleta
Cargar modelo horizontal sin canaleta
Cargar modelo vertical con canaleta
Cargar modelo vertical sin canaleta
3
2

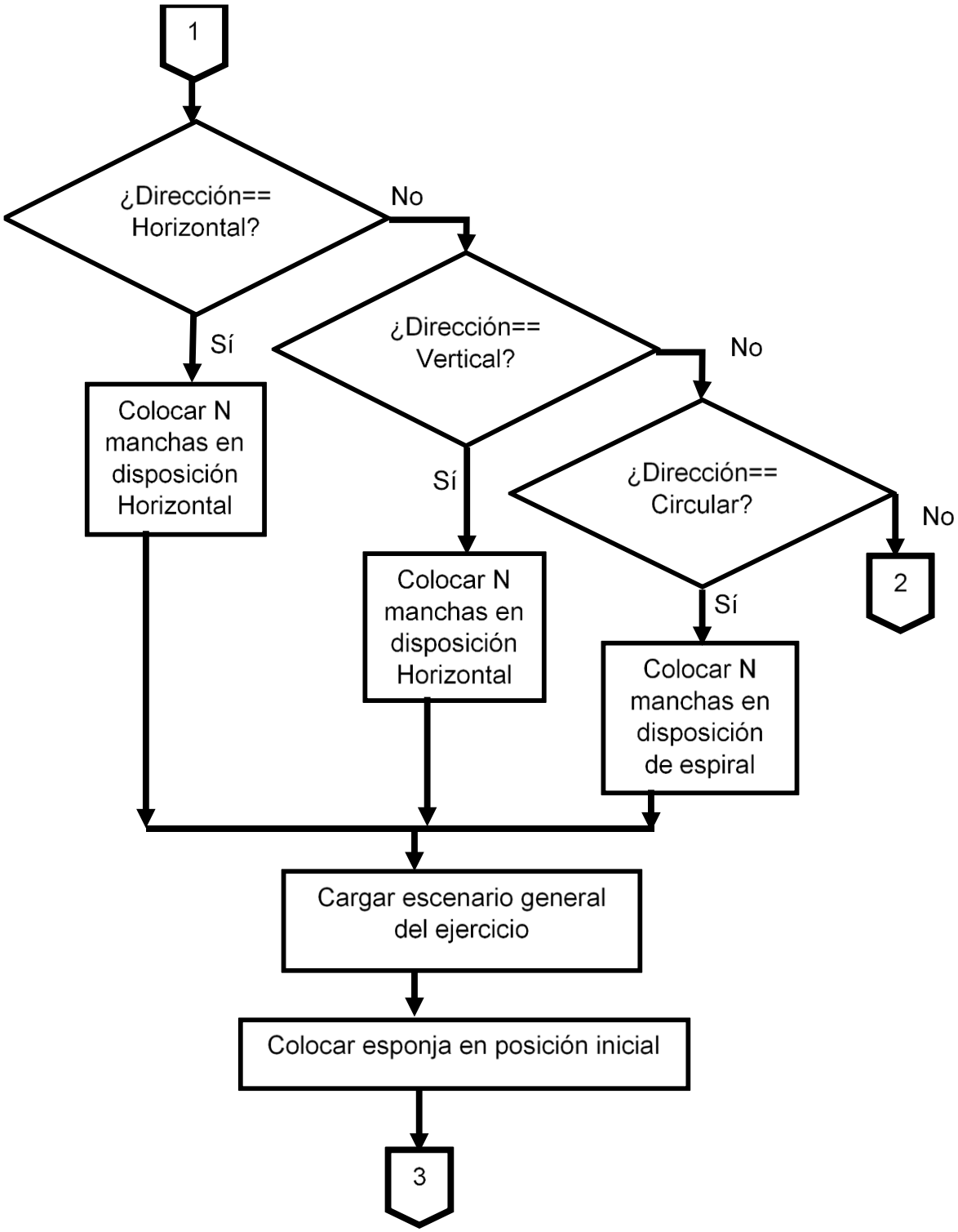

2
1
5
No. De errores=0
Colocar pelota en posición inicial
6
¿Se detecta el cuerpo del usuario?
No
Sí
¿La mano derecha del usuario está sobre la pelota?
No
Sí
Posición pelota=Posición mano derecha
4
3
¿Dificultad== normal?
No
Sí
Cargar modelo combinado con canaleta
Cargar modelo combinado sin canaleta
1

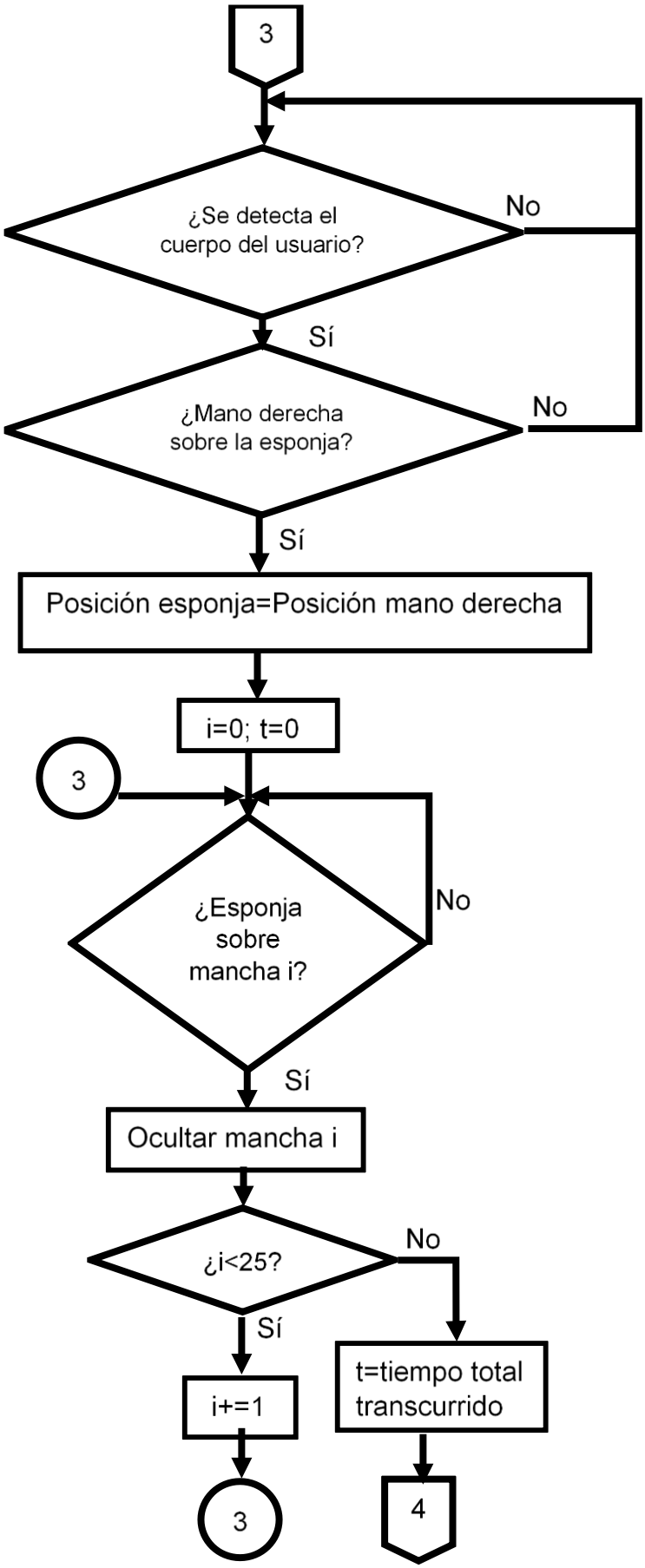


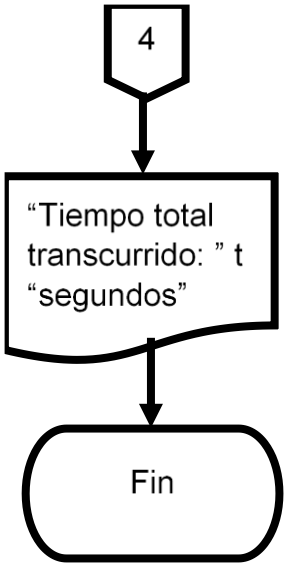
Los scripts en lenguaje C# correspondientes al programa de minigolf para sensor *Kinect* se encuentran en el apéndice 3.

-Programa de limpiar la ventana, versión para sensor *Kinect One*:




1
¿Dirección== Horizontal?
No
Sí
¿Dirección== Vertical?
No
Sí
¿Dirección== Circular?
No
Sí
Colocar N manchas en disposición Horizontal
Colocar N manchas en disposición Horizontal
Colocar N manchas en disposición de espiral
2
Cargar escenario general del ejercicio
Colocar esponja en posición inicial
3


3
¿Se detecta el cuerpo del usuario?
No
Sí
¿Mano derecha sobre la esponja?
No
Sí
Posición esponja=Posición mano derecha
i=0; t=0
3
¿Esponja sobre mancha i?
No
Sí
Ocultar mancha i
¿i<25?
No
Sí
i+=1
t=tiempo total transcurrido
3
4



Los scripts en lenguaje C#, correspondientes al programa de limpiar la ventana para sensor *Kinect* se encuentran en el apéndice 4.

7.4 Diagrama de Gantt de los entregables

Se realizaron en total nueve entregables a lo largo del desarrollo de los cuatro programas. En la figura 7.4.1 se muestra el diagrama de Gantt y en la 7.4.2 se muestran las indicaciones correspondientes.

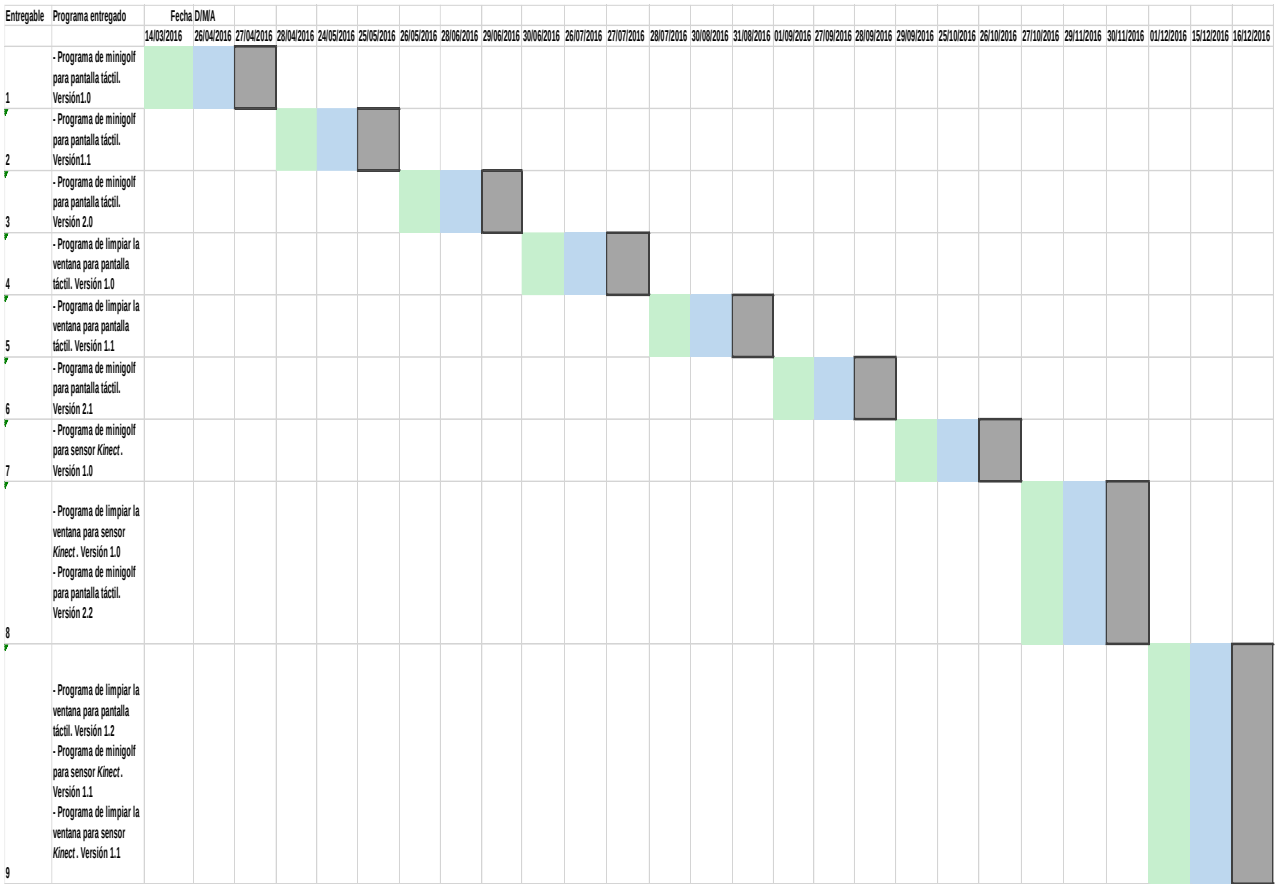


Figura 7.4.1 Diagrama de Gantt

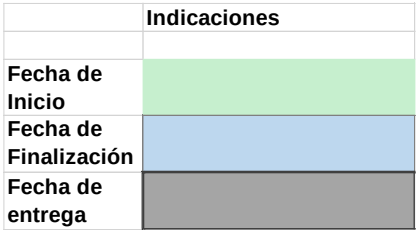


Figura 7.4.2 Indicaciones

7.5 Control de versiones de los programas

Se indica a continuación los cambios y características de cada versión de los programas desarrollados.

Versiones para pantalla táctil

Nombre del Programa	Versión	Dispositivo de entrada	Características	Cambios realizados respecto a la versión anterior
Minigolf	1.0	Pantalla táctil	- Cuenta con cuatro modos diferentes de uso: horizontal 1, horizontal 2, vertical y combinado -Únicamente existe una sola canaleta por cada modo - Utiliza métodos nativos de Unity3D para el manejo de los toques sobre la pantalla táctil - Cada choque con la canaleta se cuenta como un error - Utiliza iluminación global en tiempo real, una luz direccional en tiempo real y no se permite la proyección de sombras	No aplica
Minigolf	1.1	Pantalla táctil	-Cuenta con ocho modos diferentes de uso: horizontal 1 normal y en espejo, horizontal 1 normal y en espejo, vertical normal y en espejo, y combinado normal y en espejo	- Se agregaron cuatro modos nuevos, que consisten en el modo “en espejo” de los cuatro modos anteriores - Se indican los choques con las canaletas cambiando el color de la pelota de blanco a rojo
Minigolf	2.0	Pantalla táctil	- Cuenta con seis modos diferentes de uso: horizontal en dificultad con canaleta, horizontal en dificultad sin canaleta, vertical en dificultad con canaleta, vertical en dificultad sin canaleta, combinado en dificultad con canaleta y combinado en dificultad sin canaleta	- Se desecharon los ocho modos anteriores, por los seis nuevos
Minigolf	2.1	Pantalla táctil	- Cuenta con los mismos seis modos diferentes de uso de la versión 2.0 pero colocando múltiples canaletas por cada modo	- Se añadieron múltiples canaletas por cada modo de uso

Minigolf	2.2	Pantalla táctil	- Se mantienen los seis modos de uso de la versión 2.1 - Se utiliza el *plugin* unity3d-tuio para el manejo de los toques sobre la pantalla táctil - Utiliza iluminación global *baked*, una luz direccional *baked* y tiene sombras en modo *soft shadows* activadas	- Ahora se manejan los toques sobre la pantalla táctil con un *plugin* externo, lo que permite una correcta detección de hasta dos puntos independientes sobre la misma - Se optimizaron los efectos visuales para incrementar el rendimiento gráfico del programa - Se verifica si la pelota está debajo del área de toque de alguno de los dos puntos detectados antes de moverla, evitando así que la pelota quede atascada en alguna de las canaletas
Limpiar la ventana	1.0	Pantalla táctil	- Cuenta con tres modos de uso: horizontal, vertical y libre - Utiliza métodos nativos de Unity3D para el manejo de los toques sobre la pantalla táctil - Utiliza iluminación global en tiempo real, una luz direccional en tiempo real y no se permite la proyección de sombras	No aplica
Limpiar la ventana	1.1	Pantalla táctil	- Cuenta con tres modos de uso: horizontal, vertical y circular	- Se cambió el modo libre por un modo circular, en el cual se colocan las manchas de tal forma que parezcan una espiral - Ahora se cuenta el tiempo que toma limpiar todas las manchas de suciedad
Limpiar la ventana	1.2	Pantalla táctil	- Se utiliza el *plugin* unity3d-tuio para el manejo de los toques sobre la pantalla táctil -Se utilizan modelos nuevos para la ventana en modo en círculos y para la esponja para limpiar , en todos los modos - Utiliza iluminación global *baked*, una luz direccional *baked* y tiene sombras en modo *soft shadows* activadas	- Ahora se manejan los toques sobre la pantalla táctil con un *plugin* externo, lo que permite una correcta detección de hasta dos puntos independientes sobre la misma - Se optimizaron los efectos visuales para incrementar el rendimiento gráfico del programa - El modelo de la esponja para limpiar es más realista respecto a la versión anterior - La ventana para el modo en círculos ahora es de forma circular, en lugar de colocar las manchas en una forma de espiral

Versiones para sensor *Kinect*

Nombre del Programa	Versión	Dispositivo de entrada	Características	Cambios realizados respecto a la versión anterior
Minigolf	1.0	Sensor *Kinect*	- Está basada en la versión 2.1 del programa de minigolf para pantalla táctil, por lo cual conserva todas sus características excepto el manejo de toques sobre la pantalla táctil - La pelota se mueve respecto a la posición de la mano derecha del usuario detectado con el sensor *Kinect*	No aplica
Minigolf	1.1	Sensor *Kinect*	- Utiliza iluminación global *baked*, una luz direccional *baked* y tiene sombras en modo *soft shadows* activadas	- Se verifica si la pelota está debajo de la mano derecha del usuario antes de moverla, cuando en la versión 1.0, la pelota literalmente "seguía" a la mano derecha del usuario
Limpiar la ventana	1.0	Sensor *Kinect*	- Está basada en la versión 1.1 del programa de limpiar la ventana para pantalla táctil, por lo cual conserva todas sus características excepto el manejo de toques sobre la pantalla táctil - El trapo se mueve respecto a la posición de la mano derecha del usuario detectado con el sensor *Kinect*	No aplica
Limpiar la ventana	1.1	Sensor *Kinect*	-Se utilizan modelos nuevos para la ventana en modo en círculos y para la esponja para limpiar , en todos los modos - Utiliza iluminación global *baked*, una luz direccional *baked* y tiene sombras en modo *soft shadows* activadas	- Se optimizaron los efectos visuales para incrementar el rendimiento gráfico del programa - El modelo de la esponja para limpiar es más realista respecto a la versión anterior - La ventana para el modo en círculos ahora es de forma circular, en lugar de colocar las manchas en una forma de espiral

8. Operación del sistema

En el presente desarrollo, se crearon cuatro programas ejecutables individuales. Dos de ellos utilizan como entrada un sensor *Kinect One* y dos de ellos la pantalla táctil que se mencionó anteriormente. Una vez se inicia el programa no se requiere ninguna configuración adicional, pero si se utiliza por ejemplo la versión de minigolf para *Kinect* y se intenta utilizar con la pantalla táctil, sencillamente no responderá al dispositivo actual de entrada. De la misma forma con la versión respectiva para pantalla táctil.

8.1 Operación del programa de minigolf para pantalla táctil y sensor *Kinect*

1) Para utilizar el programa de minigolf con la pantalla táctil, deberá dar doble *click* sobre el programa ejecutable de nombre “minigolf_tactil.exe”. Si se desea utilizar el programa con el sensor *Kinect*, entonces deberá dar doble *click* sobre el programa ejecutable de nombre “minigolf_kinect.exe”.
   Al hacer esto se abrirá un pequeño rectángulo como el que se muestra en la figura 8.1.1 (en ambos casos).

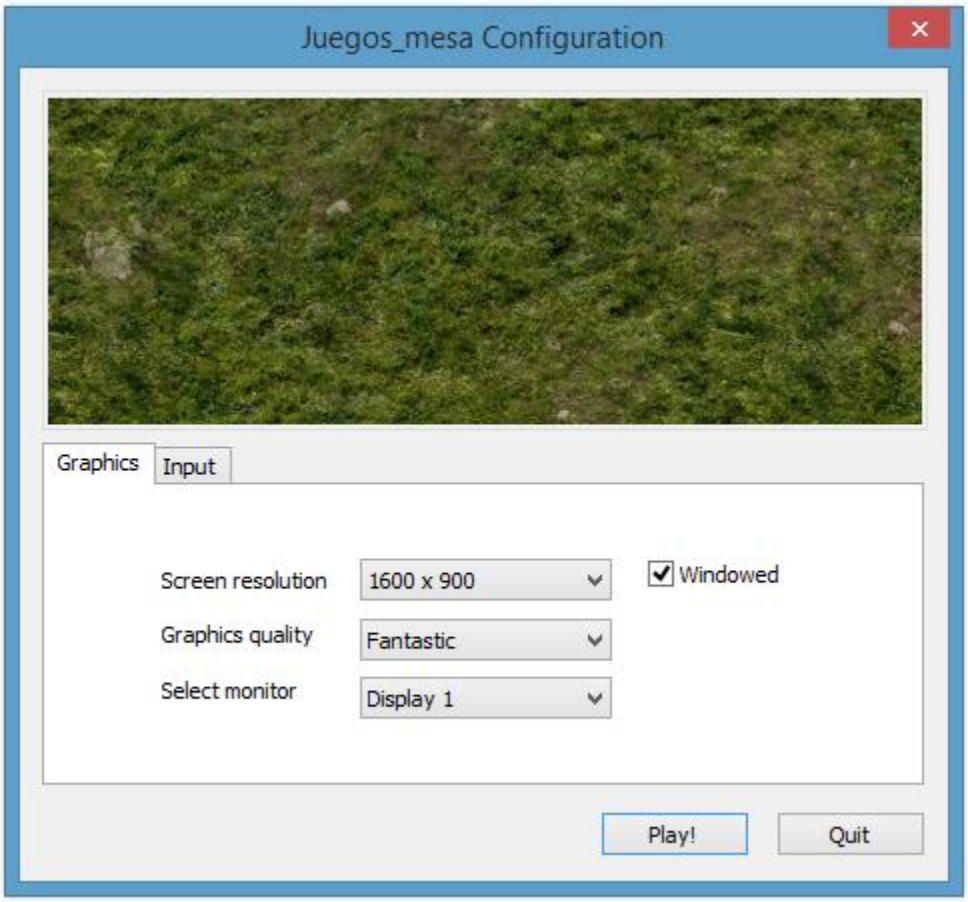


Figura 8.1.1 Configuraciones iniciales del programa

La pestaña *Graphics* o gráficos, nos permite configurar 4 parámetros: Resolución de pantalla (*Screen resolution*), Calidad de gráficos (*Graphics quality*), Monitor seleccionado (*Select monitor*), y *Windowed* (modo de ventana activo o inactivo)

-Resolución de pantalla: se refiere al tamaño de imagen que producirá el programa sobre el dispositivo de despliegue actual. De manera automática se detectan las resoluciones de pantalla compatibles y se muestran en una lista cuando se da *click* sobre el valor actual (1600x900 en el caso de la figura 8.1.1). Para la mejor calidad se deberá seleccionar el valor más alto posible de la lista.

-Calidad de gráficos: hace referencia a la calidad de los efectos visuales que tendrá el programa al ejecutarse. Idealmente se debería seleccionar el modo *Fantastic* que es el modo de máxima calidad.

-Monitor seleccionado: nos permite seleccionar el dispositivo de despliegue que se utilizará de los que se encuentran conectados a la computadora.

-Modo de ventana o *Windowed*: si se encuentra inactivo el programa se ejecutará a pantalla completa, de estar activo el programa se ejecuta sobre una ventana dentro del escritorio actual de la computadora.

Nota sobre el parámetro “Resolución de pantalla” y “Calidad de gráficos”:
Estos dos parámetros influyen en gran medida en el rendimiento del programa en ejecución. A mayor resolución de pantalla y mayor calidad de gráficos, el programa exigirá más recursos de la computadora, y si los mismos no fueran suficientes o adecuados, se observarían caídas en el *framerate* del programa. En dicho caso lo mejor sería reducir primero la calidad de gráficos hasta obtener resultados favorables. El modo *Fantastic* es el modo de máxima calidad y como tal exigirá tanto una CPU como una aceleradora gráfica adecuada, el modo *Simple* es en general el modo más adecuado para la mayoría de sistemas en que el *hardware* es limitado o sencillo, mientras que el modo *Fastest* o más rápido, es el modo que se utilizaría en caso de que la computadora cuente con *hardware* obsoleto o muy limitado.

En el caso de la pestaña *Input*, esta nos permite configurar los dispositivos de entrada como el teclado, *mouse* y *joysticks* conectados a la computadora. Sin embargo, en el caso particular de este y el resto de los ejercicios no se recomienda modificar ningún parámetro puesto que los programas únicamente hacen uso del teclado y el *mouse* para las configuraciones iniciales y algunos controles del programa principal, y el dispositivo que en realidad interactuará con el mismo será la pantalla táctil o el *Kinect One*.

2) Al iniciar el programa, este comenzará mostrando la marca de agua de Unity *Personal Edition*, tras unos segundos se mostrará la pantalla de ingreso al programa.

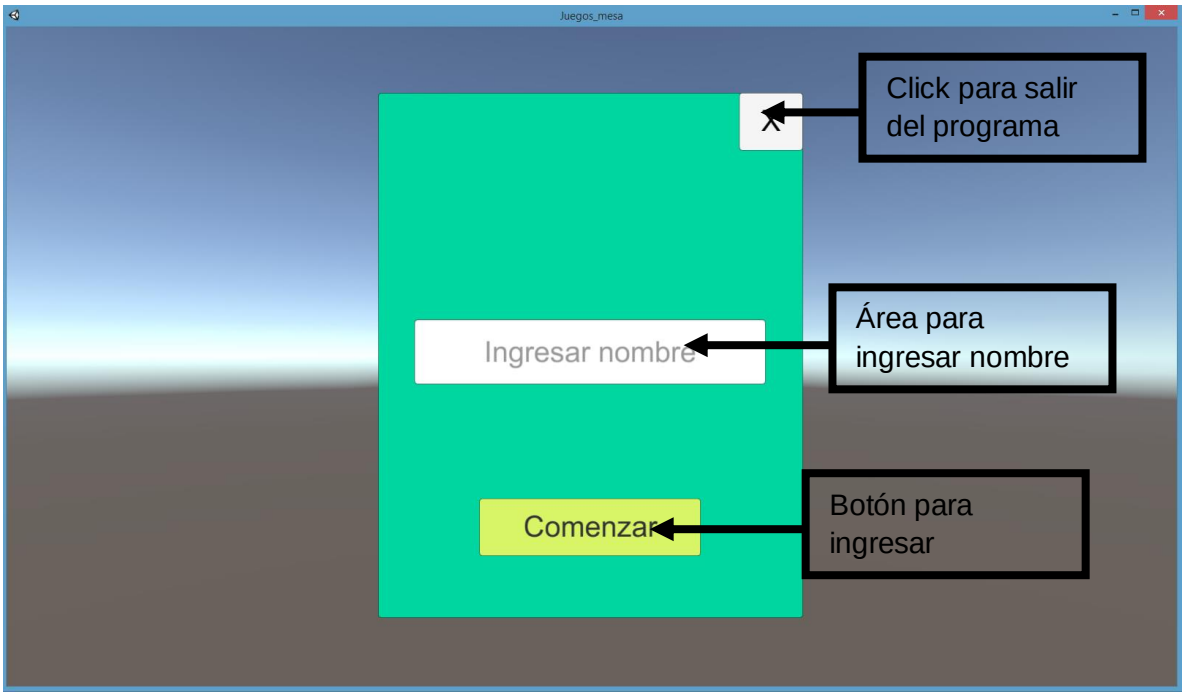


Figura 8.1.2 Pantalla de ingreso al programa.

Para ingresar simplemente haga *click* izquierdo sobre el rectángulo que indica “Ingresar nombre” y utilice el teclado para poner un nombre, a continuación deberá dar *click* izquierdo sobre el rectángulo amarillo o bien presionar la tecla “Enter” para pasar a la siguiente pantalla del programa. Si pulsa la tecla *Enter* o da *click* sobre el área marcada con el texto “Comenzar” y no se ha ingresado nada en el área marcada “Ingresar nombre” el programa no le permitirá pasar a la siguiente pantalla.

3) Una vez se ha ingresado, se mostrará la pantalla de selección del modo del ejercicio y la dificultad. Tal como se aprecia en la figura 8.1.3

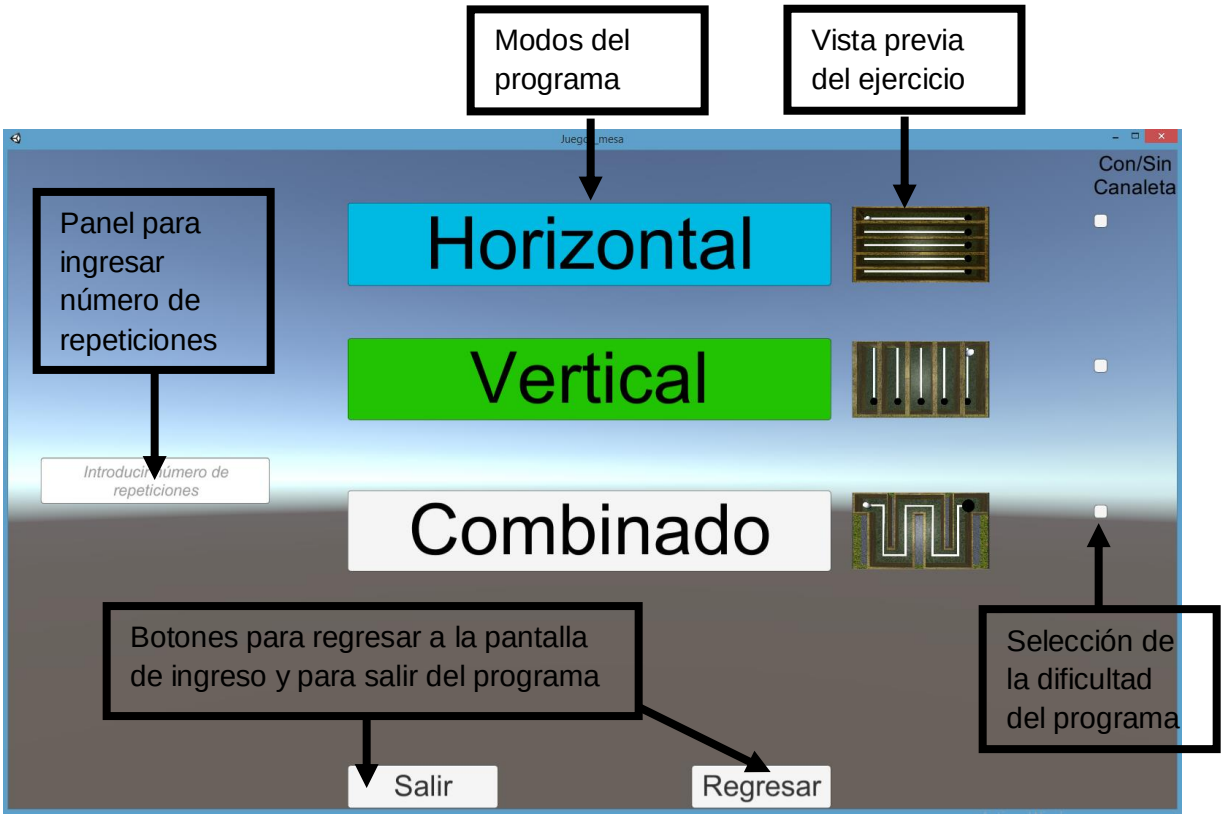


Figura 8.1.3 Pantalla de selección de modo y dificultad del programa de minigolf

Los modos disponibles son “Horizontal”, “Vertical” y “Combinado”. Los modos de dificultad son “Con” y “Sin canaleta”.

Para poder iniciar el ejercicio en el modo y dificultad elegidas deberá ingresarse el número de repeticiones que se desean utilizar. Solo se aceptarán números enteros mayores a cero o de lo contrario el programa no permitirá iniciar el ejercicio. En las imágenes situadas a la derecha de los botones de selección del modo de juego, se puede observar una vista previa del ejercicio dependiendo de si se seleccionó la dificultad Con o Sin canaleta. Para cambiar entre “Con” y “Sin canaleta” se deberá hacer *click* izquierdo sobre los pequeños cuadrados situados debajo de la leyenda “Con/Sin Canaleta”. Los cuadrados indicarán con una palomita cuando están en modo “Sin canaleta” o permanecerán en blanco si está seleccionado el modo “Con Canaleta”.

4) Tras seleccionar un modo y dificultad para el juego y haber ingresado un número de repeticiones determinado, se iniciará el juego con los parámetros establecidos. Tal como se aprecia en las figuras 8.1.4, 8.1.5, 8.1.6, 8.1.7, 8.1.8 y 8.1.9, en

cualquiera de los modos y dificultades seleccionados, la interfaz tiene el mismo funcionamiento.

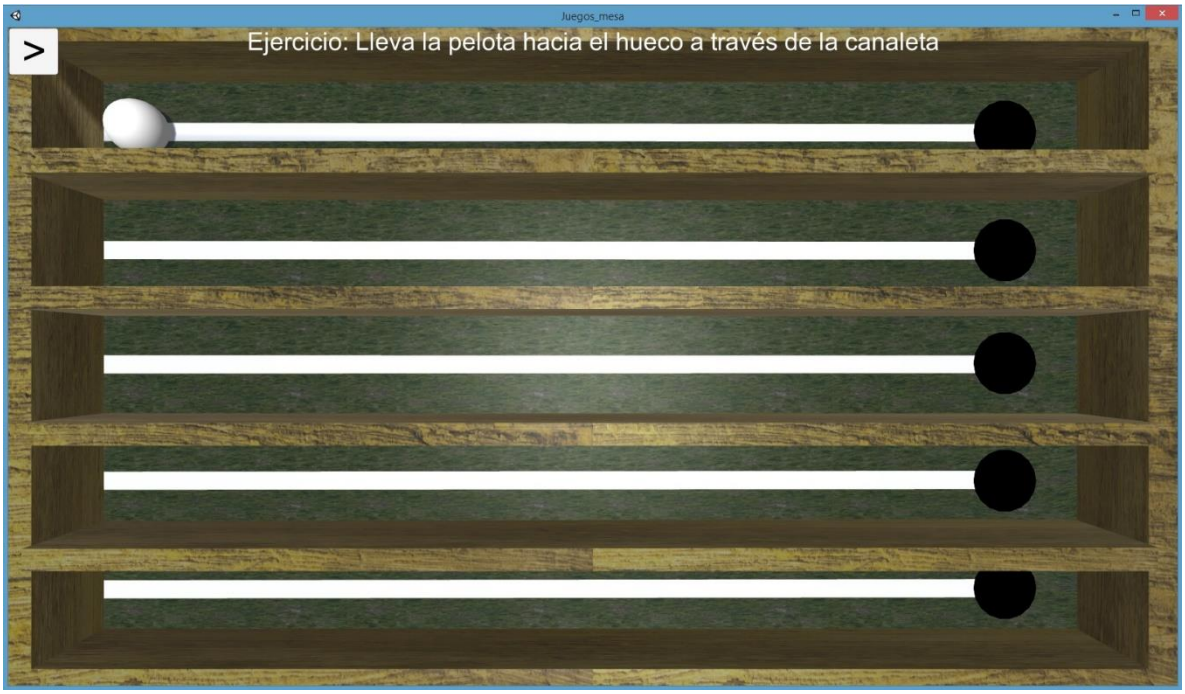


Figura 8.1.4 Programa ejecutándose en modo horizontal y dificultad con canaleta

Figura 8.1.5 Programa ejecutándose en modo horizontal y dificultad sin canaleta

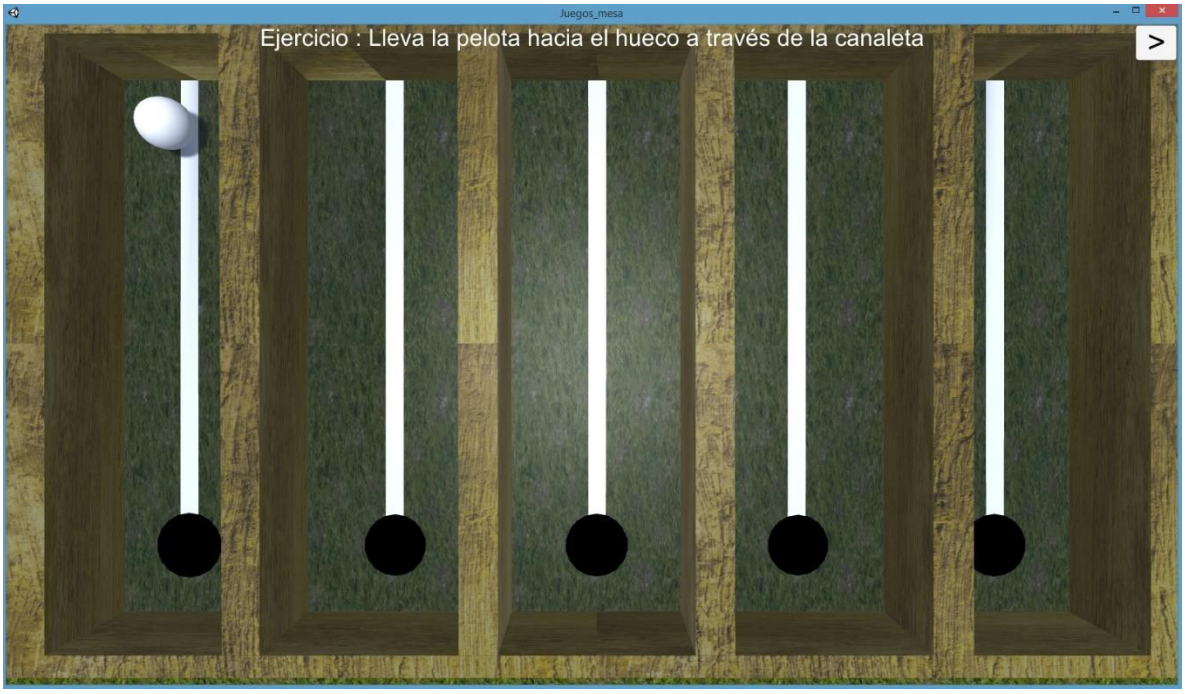


Figura 8.1.6 Programa ejecutándose en modo vertical y dificultad con canaleta

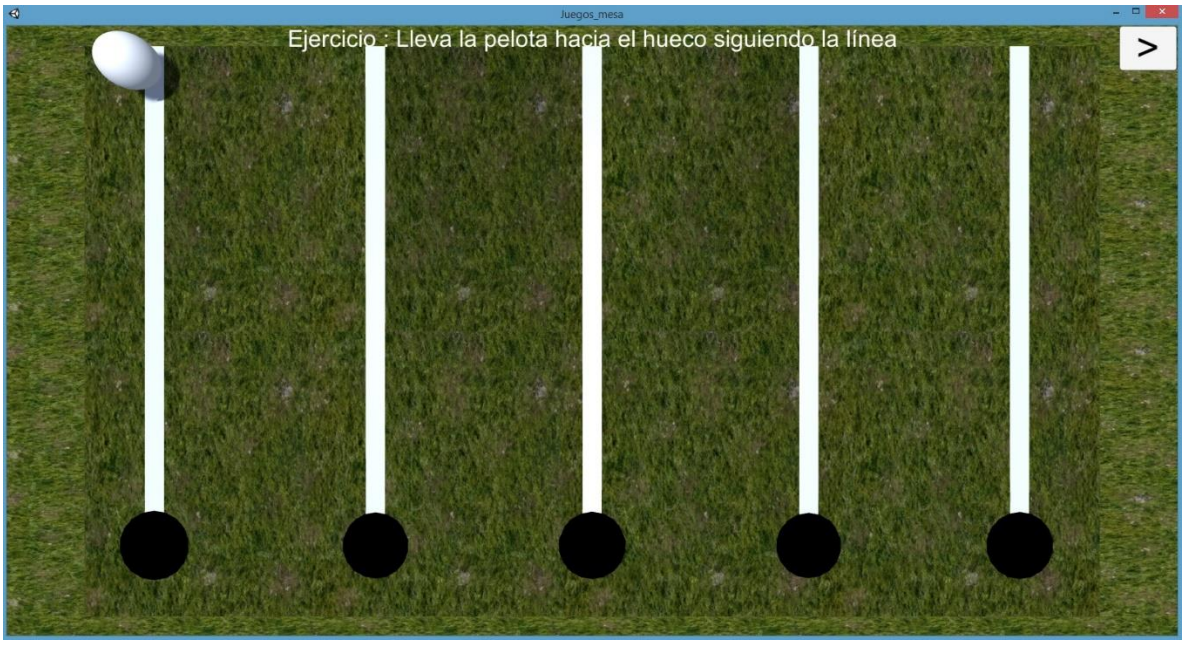


Figura 8.1.7 Programa ejecutándose en modo vertical y dificultad sin canaleta

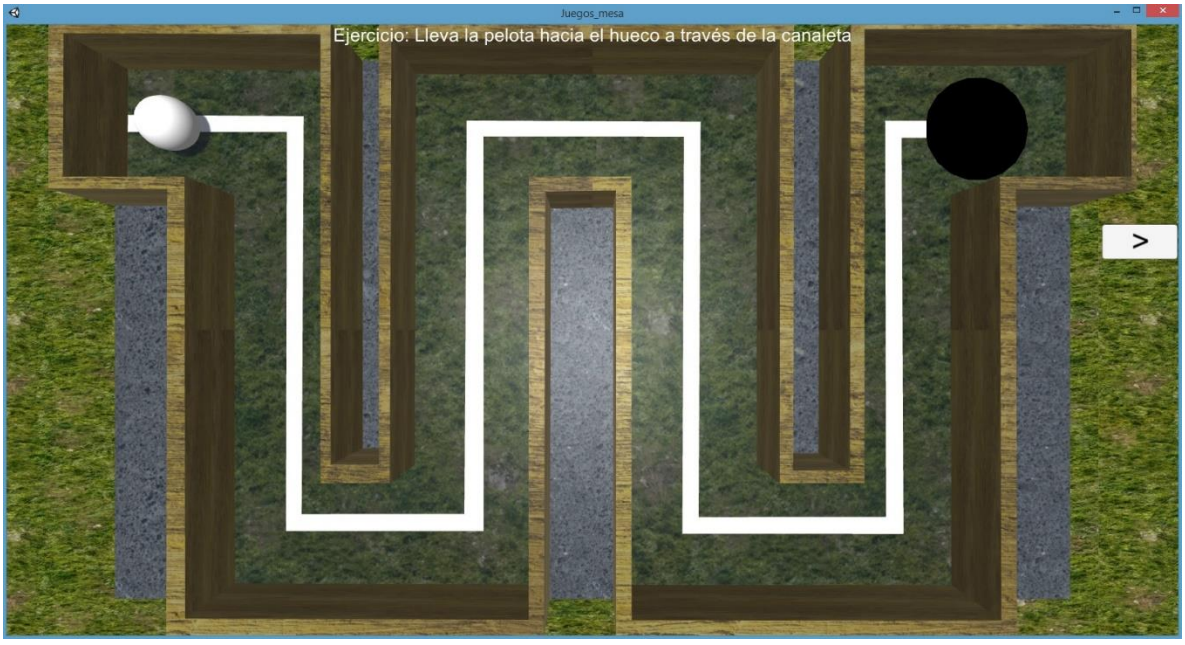


Figura 8.1.8 Programa ejecutándose en modo combinado y dificultad con canaleta

Figura 8.1.9 Programa ejecutándose en modo combinado y dificultad sin canaleta

Se muestra en la parte superior una indicación breve de lo que se debe realizar. Así mismo, se muestra un símbolo “>” sobre un botón. Este botón se puede presionar para desplegar los botones “Salir”, “Reinicio” y “Menú” tal como se observa en la figura 8.1.10. El botón salir nos permite cerrar el programa. Mientras que el botón reinicio nos permite comenzar las repeticiones desde el principio, y el botón menú nos permite regresar a la pantalla de configuraciones de la figura 8.1.3. Este funcionamiento es válido en cualquier modo y dificultad del ejercicio. Únicamente existirán variaciones en cuanto a la posición de la pantalla en que se muestra el botón “>”. Esto se hizo con el fin de estorbar lo menos posible a la vista cuando se realizará el ejercicio para cada caso particular.

Figura 8.1.10 Programa ejecutándose en modo horizontal y dificultad con canaleta con botones desplegados

El programa reaccionará inmediatamente a las entradas táctiles sobre la pantalla (si se ejecutó el programa “minigolf_tactil.exe”) o bien a los movimientos del usuario a través del sensor *Kinect* (si se ejecutó el programa “minigolf_kinect.exe”). Una vez se realicen las repeticiones configuradas en la pantalla mostrada en la figura 8.1.3, el programa mostrará la puntuación final, tal como se observa en la figura 8.1.11.

Figura 8.1.11 Programa finalizado mostrando la puntuación final por repetición

Se cuentan los errores por repetición así como los errores totales. Si se desea realizar de nuevo el ejercicio se deberá dar *click* en el botón “Reiniciar” o bien “Menú” si se desea utilizar el ejercicio con un modo, dificultad y/o número de repeticiones diferente.

8.2 Operación del programa de limpiar la ventana para pantalla táctil y sensor *Kinect*

1) Para utilizar el programa con la pantalla táctil, deberá dar doble *click* sobre el programa ejecutable con nombre “limpiarventana_tactil.exe”. Si desea utilizar el programa con el sensor *Kinect*, entonces deberá dar doble *click* sobre el programa ejecutable con nombre “limpiarventana_kinect.exe”. Al hacer esto se abrirá un pequeño rectángulo como el que se muestra en la figura 8.2.1 (en ambos casos).

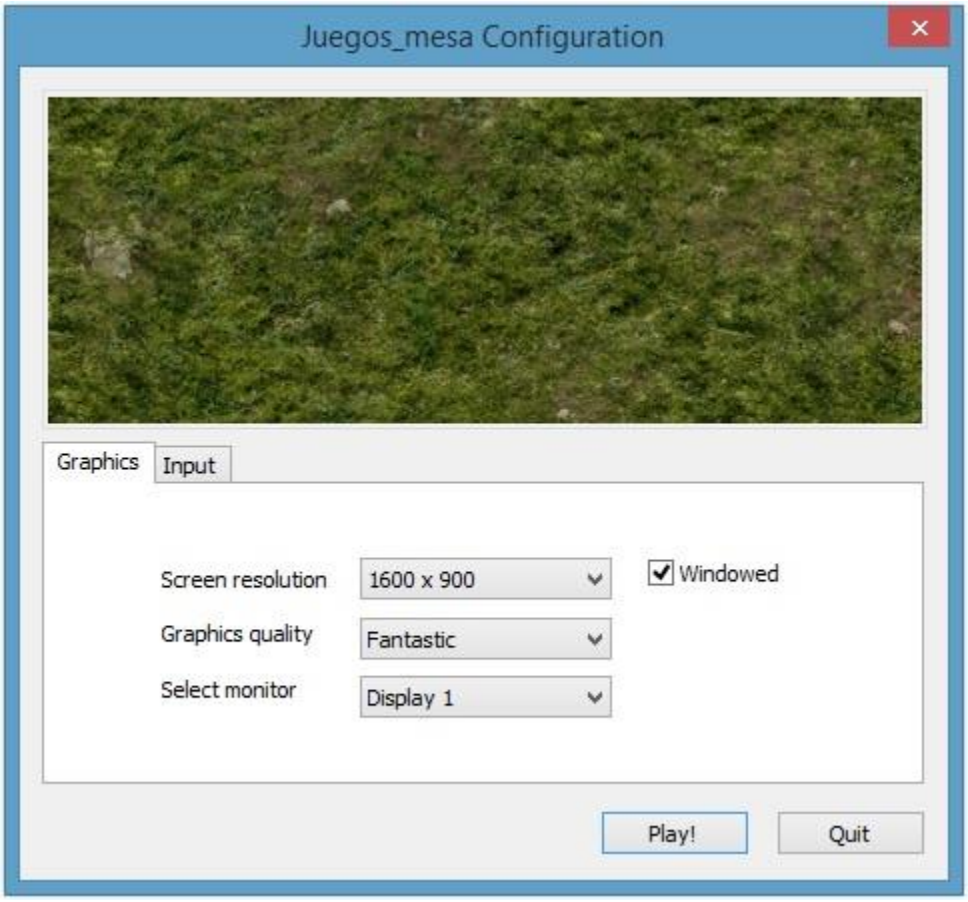


Figura 8.2.1 Pantalla de configuración del programa

Tal como en el programa de minigolf, existen 4 parámetros configurables en la pestaña de Gráficos o *Graphics.* Que son *Screen resolution* (resolución de pantalla), *Graphics quality* (calidad de gráficos), *Select monitor* (selección de monitor) y *Windowed* (modo de ventana activo o inactivo). Así como algunos parámetros configurables en la pestaña *Input*. Estos parámetros funcionan de manera idéntica a los del programa de minigolf:

-Resolución de pantalla: resolución en que se ejecutará el programa gráfico tanto en modo de pantalla completa como en modo de ventana

-Calidad de gráficos: calidad de los efectos visuales durante la ejecución del programa gráfico

-Selección de monitor: dispositivo de despliegue donde se mostrará el programa gráfico en ejecución

-Modo de ventana: si está activo se ejecuta el programa gráfico en una ventana independiente sobre el escritorio actual, si está inactivo el programa gráfico se ejecutará en modo de pantalla completa.

Los parámetros de la pestaña *Input* permiten configurar los dispositivos de entrada como el *mouse*, el teclado y *joysticks*, sin embargo como se mencionó anteriormente no es necesario ni recomendable alterar esta configuración puesto que el programa únicamente hace uso del teclado y el mouse en las partes de configuración, y en la ejecución de la parte principal del programa será el *Kinect One* o bien la pantalla táctil los dispositivos que interactuarán directamente con el sistema.

2) Una vez se configuraron los parámetros de la pantalla anterior, se mostrará la pantalla de selección de modo y número de repeticiones, tal como se observa en la figura 8.2.2

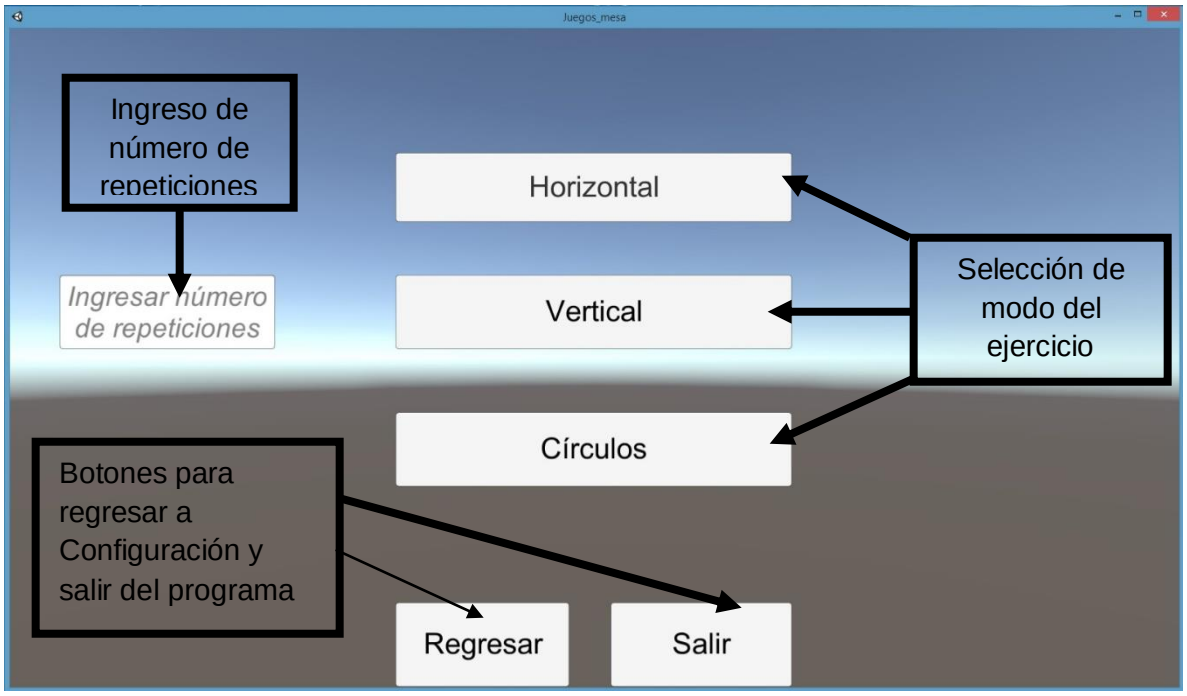


Figura 8.2.2 Pantalla de configuración de modo y número de repeticiones para el programa de limpiar la ventana

En este caso se pueden configurar tres modos distintos:

-Horizontal: las manchas deben limpiarse de izquierda a derecha

-Vertical: las manchas deben limpiarse de arriba hacia abajo

-Círculos: las manchas deben limpiarse siguiendo la espiral que forman las mismas

Antes de poder seleccionar un modo, deberá ingresarse al recuadro “Ingresar número de repeticiones” un entero mayor a cero. Este número entero definirá la cantidad de manchas presentes en el ejercicio.

3) Una vez se ingresó un número de repeticiones válido y se dio *click* en alguno de los tres modos disponibles, iniciará el ejercicio basado en las configuraciones elegidas. En las figuras 8.2.3, 8.2.4, y 8.2.5 se pueden apreciar los tres diferentes modos del ejercicio para un valor de 25 repeticiones.

Figura 8.2.3 Programa de limpiar la ventana en modo horizontal con 25 repeticiones

Figura 8.2.4 Programa de limpiar la ventana en modo vertical con 25 repeticiones

Figura 8.2.5 Programa de limpiar la ventana en modo Círculos con 25 repeticiones

Una vez que el programa está en ejecución y a partir de que se limpia la primera mancha, se comenzará a contar el tiempo en segundos mediante un cronómetro interno. Tras limpiar la última mancha, el cronómetro se detendrá y se mostrará en pantalla el tiempo total en segundos y décimas de segundo que tomó el limpiar las manchas de la ventana, tal como se observa en la figura 8.2.6
Al igual que el programa de minigolf, es posible reiniciar el ejercicio con el mismo número de repeticiones y modo actual dando *click* en el botón “Reiniciar”. También se puede volver a la pantalla de configuración de modo y número de repeticiones para elegir un número de repeticiones y/o modo distinto dando *click* en el botón “Regresar”. O bien salir del programa dando click en el botón “Salir”.

Figura 8.2.6 Programa de limpiar la ventana finalizado

8.3 Requisitos de *hardware* mínimos recomendados

Para cualquiera de los 4 programas se recomiendan las siguientes características de *hardware* para un desempeño óptimo y correcto.

-CPU de dos núcleos físicos o más, marca AMD o Intel, preferiblemente a 1.6 Ghz o más

-Memoria RAM de 4 GB, preferiblemente DDR3 o DDR4

-Aceleradora gráfica dedicada de marca AMD o Nvidia, con al menos 1 GB de memoria de video dedicada, compatible con Direct3D 11.0 o superior

*Para los programas que utilizan sensor *Kinect*:

-Sensor *Kinect* V2 (versión para *Xbox One*)

*Para los programas que utilizan pantalla táctil

-Pantalla táctil marca PQLabs con al menos 2 puntos de detección de toques

9. Resultados, conclusiones y perspectivas

9.1 Pruebas de facilidad de uso

Las siguientes pruebas tienen por objetivo evaluar la facilidad con que se pueden utilizar los programas tanto en su versión para pantalla táctil como para *Kinect One*. En el sentido de que tan intuitiva resulta la interfaz de usuario así como dificultad del uso del programa al interactuar con el mismo. Al ser estos parámetros subjetivos, no es posible medirlos utilizando alguna unidad del Sistema Internacional de Unidades, por lo cual se optó por una escala simple del 1 al 10, para después pedir sugerencias o explicaciones del motivo por el cual no se calificó con el máximo valor de la escala. No se realizaron pruebas de otro tipo puesto que se consideró que la terapeuta especialista en neurorehabilitación y la responsable del Departamento de Cómputo del laboratorio indicaron que únicamente se requerían pocos cambios en la versión preliminar, mismos que se incluyen en el apartado de “Cambios realizados tras la realización de las pruebas**”.**

Para realizar las pruebas se pidió a tres personas del laboratorio que utilizaran tanto el programa de minigolf como el de limpiar la ventana en su penúltima versión y a partir de esto se obtuvieron las versiones finales.

Programa de minigolf:

Usuario	**¿Qué tan intuitiva es la interfaz del programa? (escala del 1 al 10)**	**¿Por qué calificó el parámetro anterior con dicha puntuación?**	**¿Qué tan intuitivo es el uso del programa en la parte de los ejercicios?**	**¿Por qué calificó el parámetro anterior con dicha puntuación?**	**Observaciones**
1	9	En el inicio del programa, no es muy claro el hecho de que, primero se debe colocar el número de repeticiones y luego elegir un modo de juego, suele ser más común que sea en sentido inverso	9	En ocasiones resulta algo difícil mover la pelota puesto que pareciera que no sigue a la mano sino que aparece donde está la mano en ese momento	El programa tiene problemas cuando se coloca todo el brazo sobre la pantalla táctil
2	9	No resulta del todo claro, a que se refiere el programa con "repeticiones".	9	El movimiento de la pelota pareciera tener un ligero *delay* en su movimiento	La pelota se queda atascada en las canaletas y ya no se puede mover, lo cual te obliga a reiniciar el programa
3	9	Al principio resulta poco lógico el orden en que se configuran los parámetros de la partida. Debería existir alguna indicación o ayuda.	9	El manejo de la pelota cuando se utiliza toda la meno es un poco extraño porque alterna su posición entre diferentes dedos de la mano	La pelota se pone roja al chocar con las canaletas, pero sería bueno que además regresara a una posición correcta nuevamente en lugar de solo ponerse roja y quedarse en ese lugar

Programa de limpiar la ventana:

Usuario	¿Qué tan intuitiva es la interfaz del programa? (escala del 1 al 10)	¿Por qué calificó el parámetro anterior con dicha puntuación?	¿Qué tan intuitivo es el uso del programa en la parte de los ejercicios?	¿Por qué calificó el parámetro anterior con dicha puntuación?	Observaciones
1	9	No es tan claro en que afecta el número de repeticiones al iniciar el ejercicio	8	Debería existir un indicador o alguna ayuda visual para entender cómo debe moverse la esponja al limpiar la ventana. Las manchas se confunden un poco con el ambiente que está atrás	Las manchas se mueven sobre la pantalla pero repentinamente "desaparecen" y luego vuelven a aparecer
2	8	Igual que en el ejercicio de golf, hace falta volver a configurar todo si se quiere cambiar el número de repeticiones. En algunas aplicaciones y videojuegos el orden en que se configura la partida es indistinto, y eso es más cómodo, en este programa tienes que recordar el orden correcto en que se configuran o no puedes iniciar	8	No es claro cómo debe moverse la esponja para poder limpiar la mugre en la ventana. A veces la esponja parece no limpiar las manchas y debe moverse la esponja de nuevo sobre la mancha	El programa no trabaja bien cuando se pone todo el brazo completo sobre la pantalla táctil
3	9	Cuando se pone un número mayor a 25 se borra lo que se había escrito en la parte del número de repeticiones.	8	Es difícil al principio saber cómo mover la esponja para limpiar, en especial el modo círculos, es muy extraño y no se entiende la dirección en que debe limpiarse.	El programa podría tener problemas para ejecutarse en computadoras sin tarjeta gráfica dedicada

9.2 Pruebas de rendimiento

Programa de minigolf:

Se indica a continuación el *framerate* obtenido para los programas de minigolf en tres diferentes equipos, de tres gamas diferentes, en modo de calidad máxima (*Fantastic*) y mínima (*Fastest*), utilizando el *software* MSI Afterburner. En las figuras 9.2.1 a 9.2.6 se pueden observar capturas de pantalla de los resultados.

Tipo de equipo	**Modelo de CPU**	**Modelo de tarjeta gráfica**	***Framerate* obtenido en calidad mínima**	***Framerate* obtenido en calidad máxima**	**Resolución utilizada**	**Sistema Operativo**	**API gráfica**
Laptop	Intel Celeron 1000M a 1.8 Ghz, Dos núcleos físicos, gama baja	Intel HD Graphics (Ivy Bridge) integrada, gama baja	118.8 fps	33.9 fps	1366x768 píxeles	Windows 8.0	Direct 11.0
Desktop	Intel Core i7 6700 a 3.4 Ghz, Cuatro núcleos físicos, ocho núcleos virtuales, gama alta	NVIDIA Geforce GTX 970 dedicada, gama alta	357.1 fps	59.9 fps	1920x1080 píxeles	Windows 8.1	Direct 11.0
Desktop	Intel Core i7 4790 a 3.6 Ghz, Cuatro núcleos físicos, ocho núcleos virtuales, gama alta	AMD Radeon R9 270 dedicada, gama media	307.9 fps	59.9 fps	1920x1080 píxeles	Windows 10	Direct 11.0

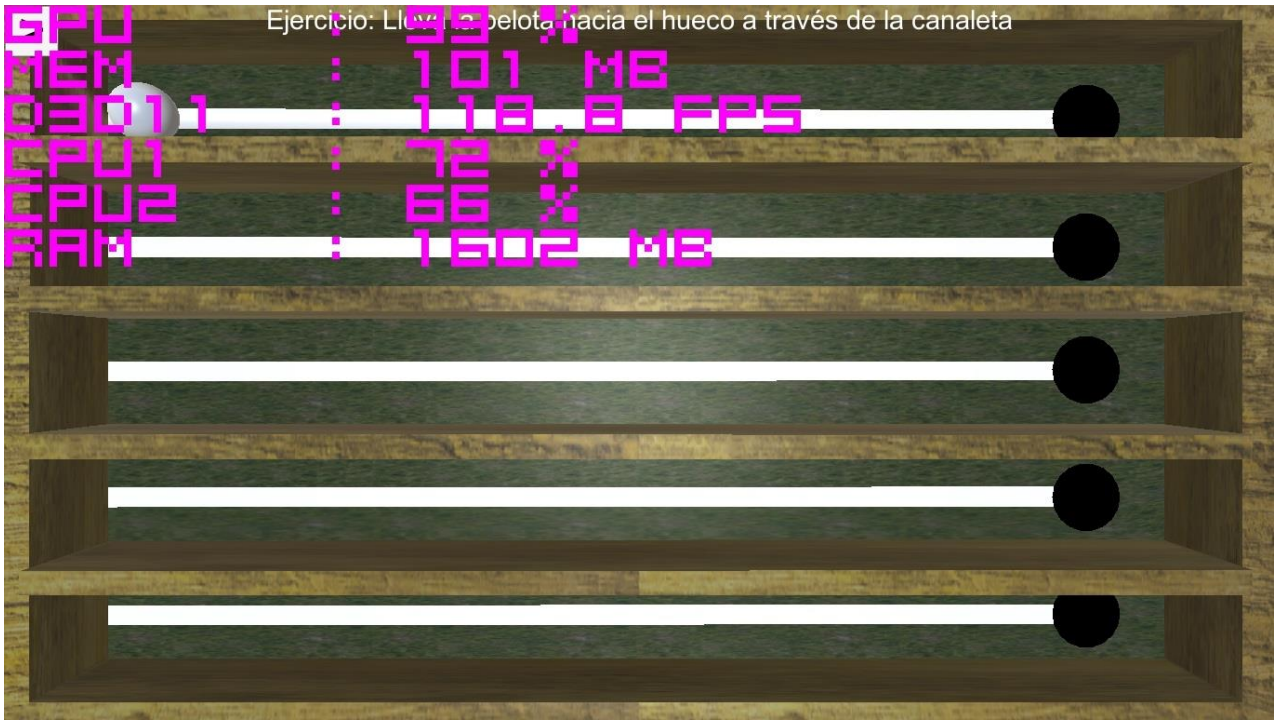


Figura 9.2.1 *Framerate* del programa de minigolf en modo *Fastest* ejecutándose en el equipo portátil

Figura 9.2.2 *Framerate* del programa de minigolf en modo *Fantastic* ejecutándose en el equipo portátil

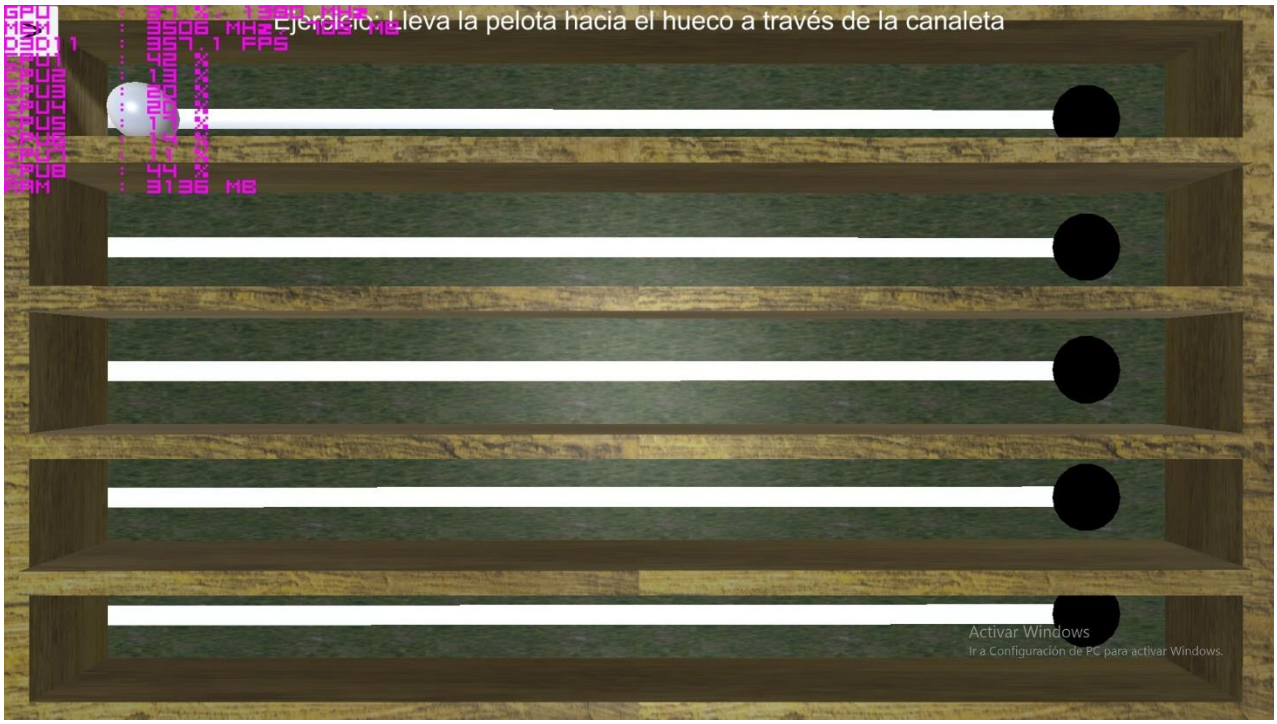


Figura 9.2.3 *Framerate* del programa de minigolf en modo *Fastest* ejecutándose en el equipo de escritorio con aceleradora gráfica Nvidia GTX 970

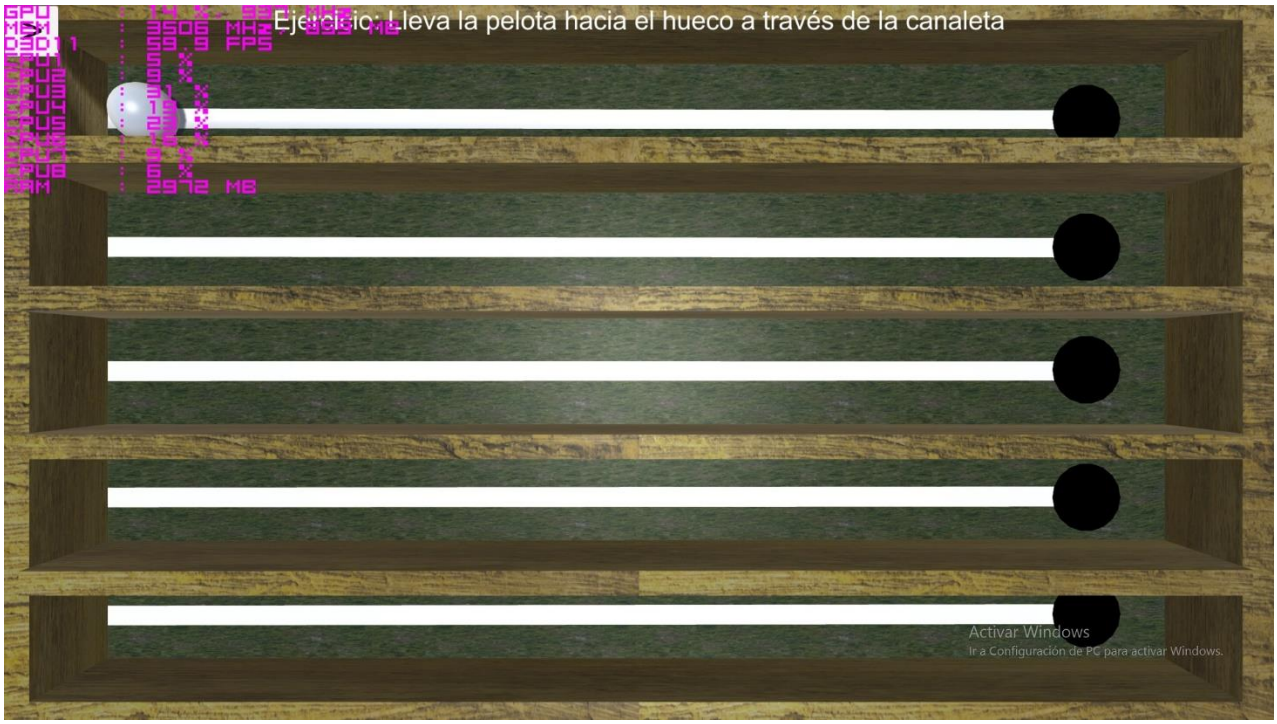


Figura 9.2.4 *Framerate* del programa de minigolf en modo *Fantastic* ejecutándose en el equipo de escritorio con aceleradora gráfica Nvidia GTX 970

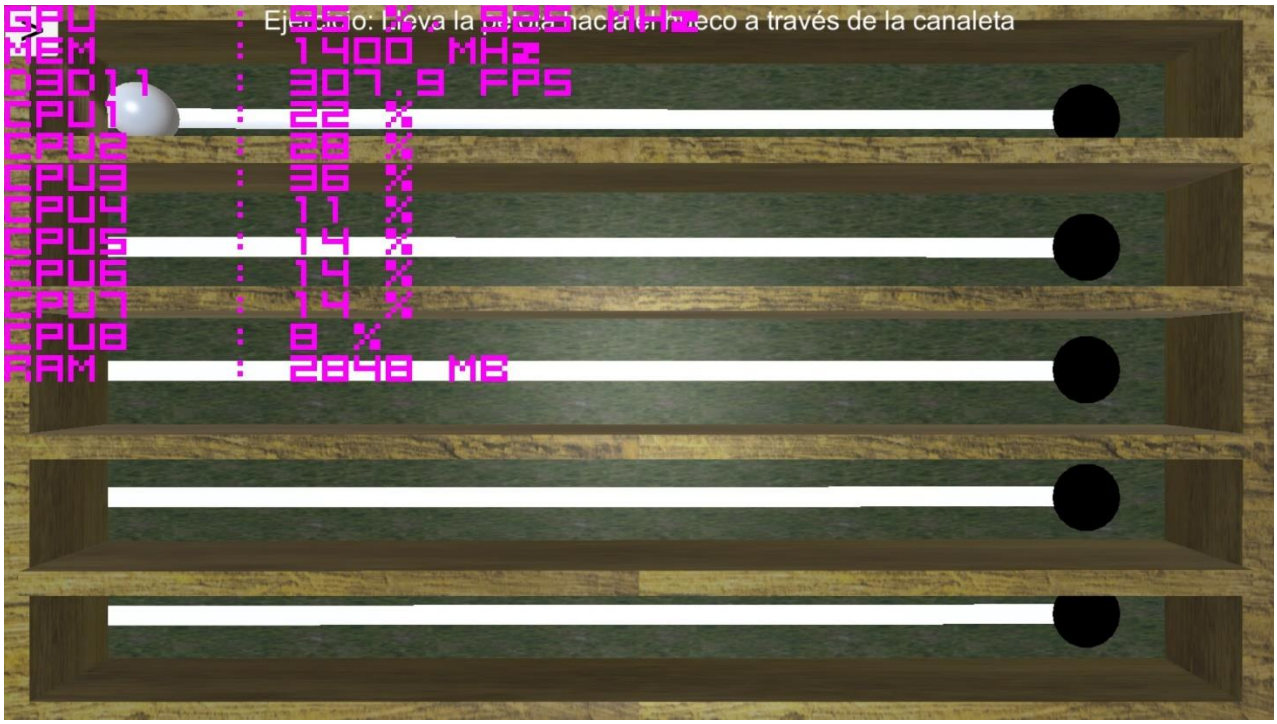


Figura 9.2.5 *Framerate* del programa de minigolf en modo *Fastest* ejecutándose en el equipo de escritorio con aceleradora gráfica AMD R9 270

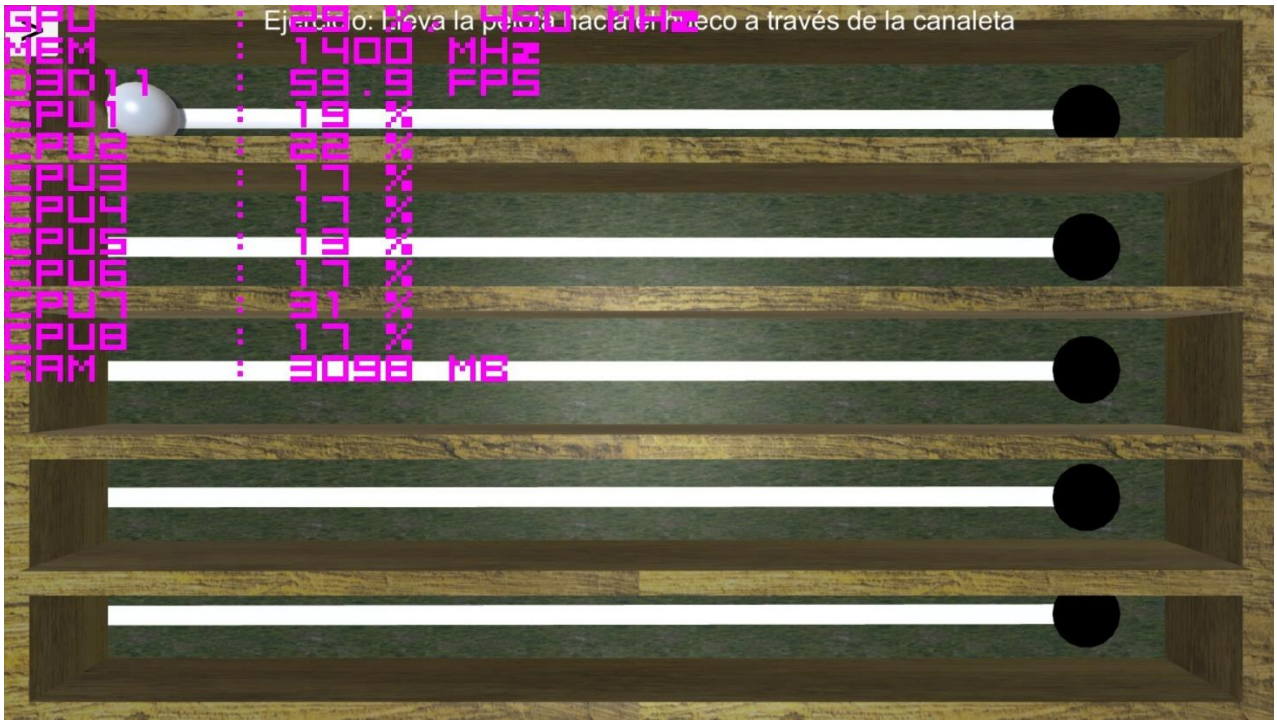


Figura 9.2.6 *Framerate* del programa de minigolf en modo *Fantastic* ejecutándose en el equipo de escritorio con aceleradora gráfica AMD R9 270

Programa de limpiar la ventana:

Se indica a continuación el *framerate* obtenido para los programas de limpiar la ventana bajo tres diferentes equipos, de tres gamas diferentes, en modo de calidad máxima (*Fantastic*) y mínima (*Fastest*), utilizando el *software* MSI Afterburner. En las figuras 9.2.7 a 9.2.12 se pueden observar capturas de pantalla de los resultados.

Tipo de equipo	**Modelo de CPU**	**Modelo de tarjeta gráfica**	***Framerate* obtenido en calidad mínima**	***Framerate* Obtenido en calidad máxima**	**Resolución utilizada**	**Sistema Operativo**	**API gráfica**
Laptop	Intel Celeron 1000M a 1.8 Ghz, Dos núcleos físicos, gama baja	Intel HD Graphics (Ivy Bridge) integrada, gama baja	27.3 fps	6.4 fps	1366x768 píxeles	Windows 8.0	Direct 11.0
Desktop	Intel Core i7 6700 a 3.4 Ghz, Cuatro núcleos físicos, ocho núcleos virtuales, gama alta	NVIDIA Geforce GTX 970 dedicada, gama alta	368.2 fps	59.9 fps	1920x1080 píxeles	Windows 8.1	Direct 11.0
Desktop	Intel Core i7 4790 a 3.6 Ghz, Cuatro núcleos físicos, ocho núcleos virtuales, gama alta	AMD Radeon R9 270 dedicada, gama media	217 fps	60 fps	1920x1080 píxeles	Windows 10	Direct 11.0

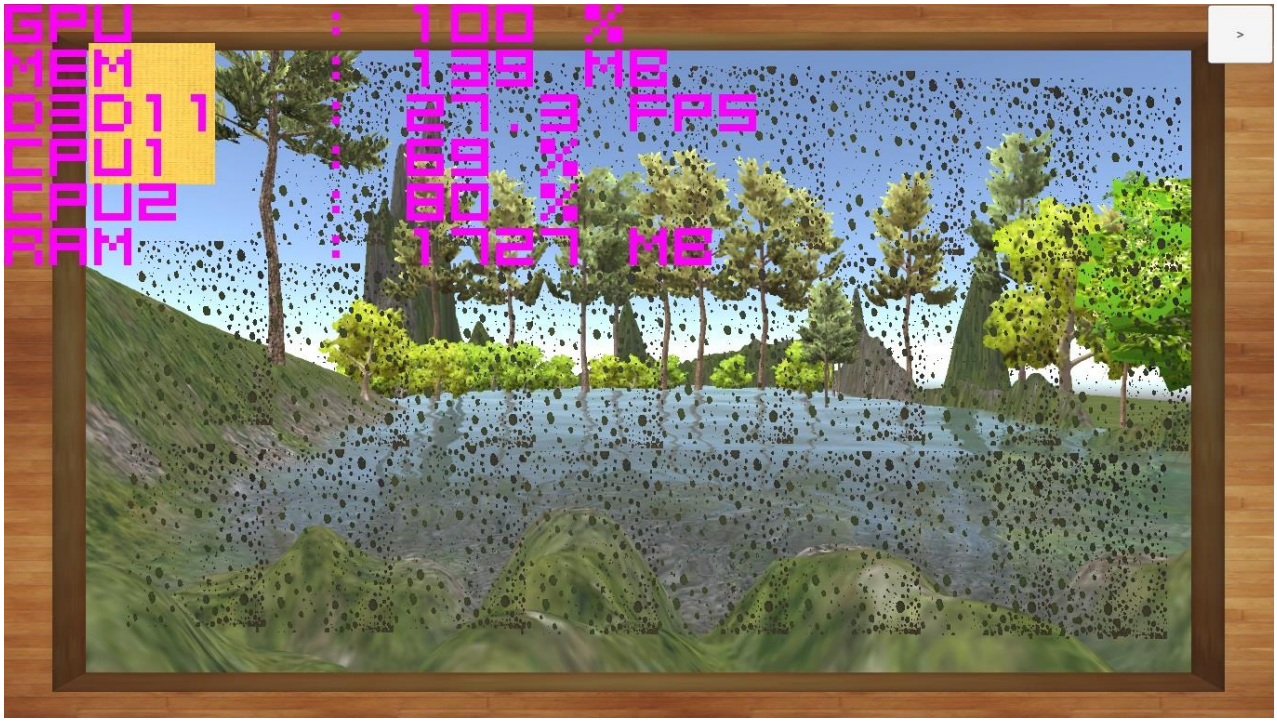


Figura 9.2.7 *Framerate* del programa de limpiar la ventana en modo *Fastest* ejecutándose en el equipo portátil

Figura 9.2.8 *Framerate* del programa de limpiar la ventana en modo *Fantastic* ejecutándose en el equipo portátil

9.2.9 *Framerate* del programa de limpiar la ventana en modo *Fastest* ejecutándose en el equipo de escritorio con aceleradora gráfica Nvidia GTX 970

9.2.10 *Framerate* del programa de limpiar la ventana en modo *Fantastic* ejecutándose en el equipo de escritorio con aceleradora gráfica Nvidia GTX 970

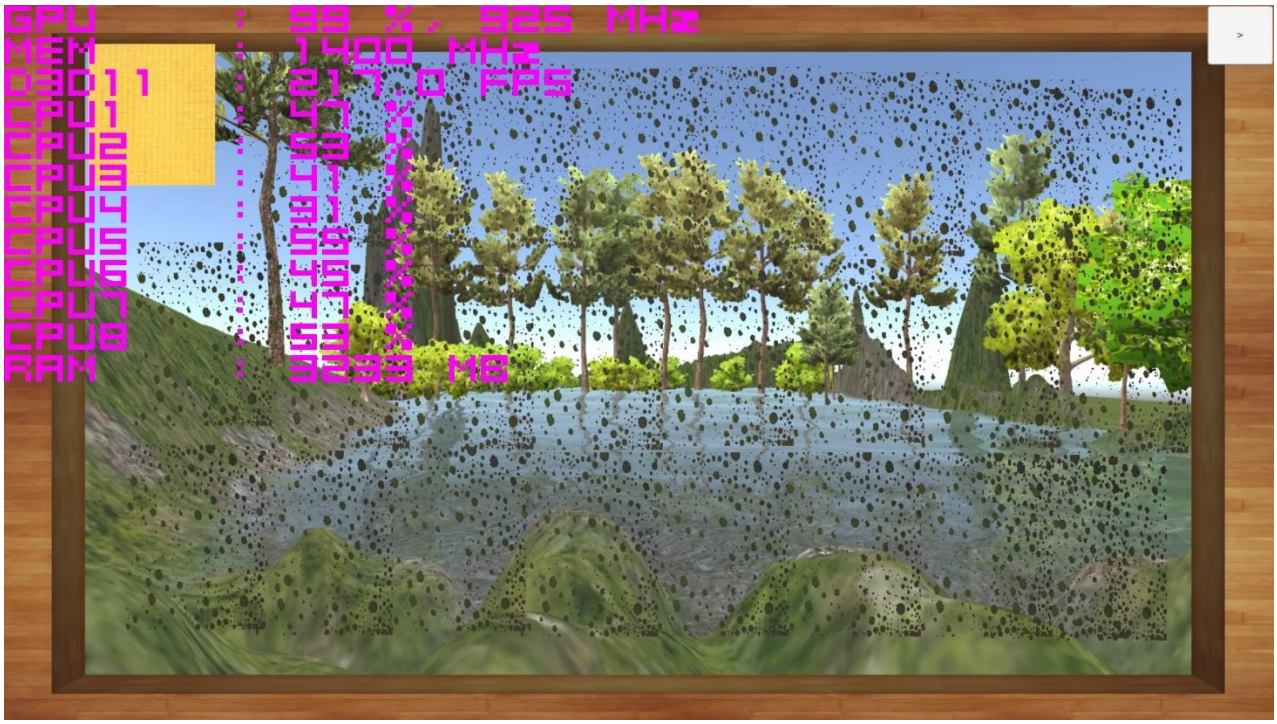


Figura 9.2.11 *Framerate* del programa de limpiar la ventana en modo *Fastest* ejecutándose en el equipo de escritorio con aceleradora gráfica AMD R9 270

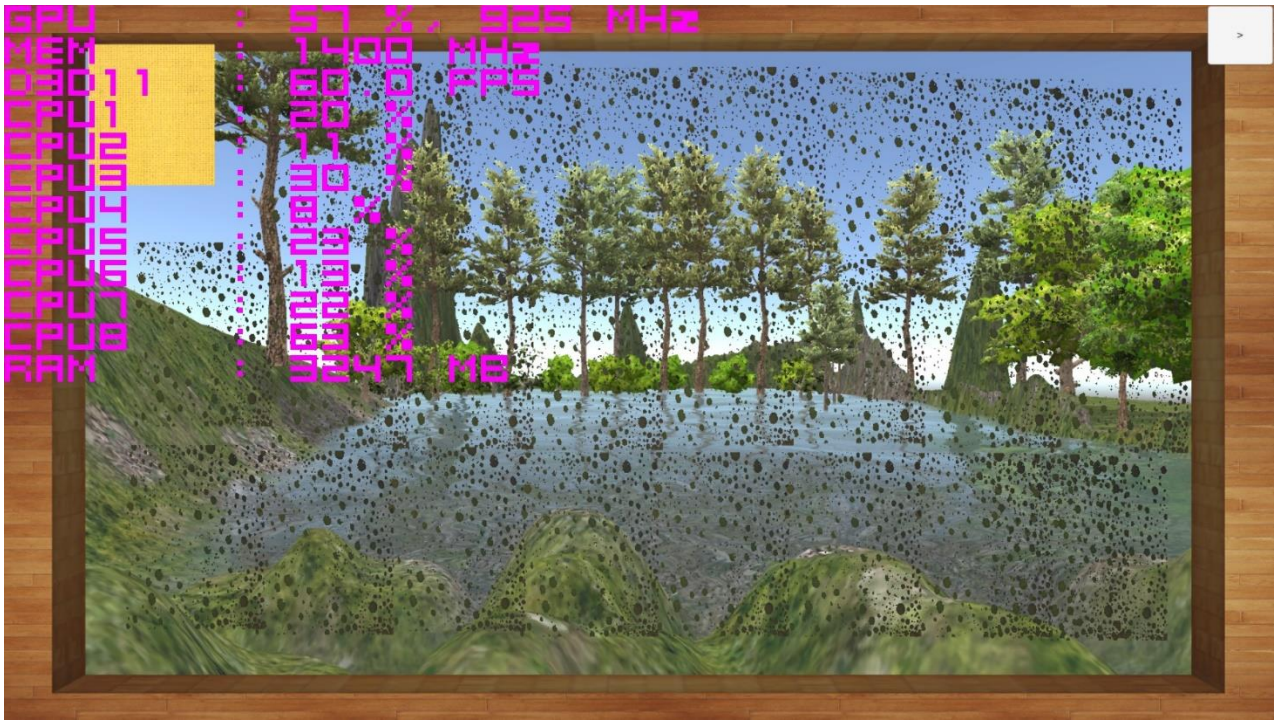


Figura 9.2.12 *Framerate* del programa de limpiar la ventana en modo *Fantastic* ejecutándose en el equipo de escritorio con aceleradora gráfica AMD R9 270

Como ya se esperaba, el programa de limpiar la ventana es el más exigente en cuanto a recursos de *hardware* se refiere, puesto que tiene problemas de *framerate* en equipos con tarjetas gráficas integradas. Tal como se explicó anteriormente, la solución es simplemente utilizar un modo de calidad más bajo, ya sea *Fastest* (el más rápido) o *Simple* (simple). Para comprobar estas afirmaciones, se realizaron dos pruebas adicionales de rendimiento, esta vez únicamente con el equipo portátil con aceleradora integrada HD graphics, en modo *Simple*, para así ver el contraste de rendimiento entre los tres modos para un equipo con hardware limitado (*Simple*, *Fastest* y *Fantastic*). Entiéndase por "*hardware* limitado", a aquella computadora que cuente con una aceleradora gráfica integrada y/o una CPU que no sea al menos de doble núcleo. En las figuras 9.2.13 y 9.2.14 se pueden observar capturas de pantalla de los resultados.

Ejercicio	**Modo de calidad**	**Resolución**	***Framerate* obtenido**
Minigolf	*Fastest*	1366x768	118.8 fps
Minigolf	*Simple*	1366x768	68.2 fps
Minigolf	*Fantastic*	1366x768	33.9 fps
Limpiar la ventana	*Fastest*	1366x768	27.3 fps
Limpiar la ventana	*Simple*	1366x768	16.1 fps
Limpiar la ventana	*Fantastic*	1366x768	6.4 fps

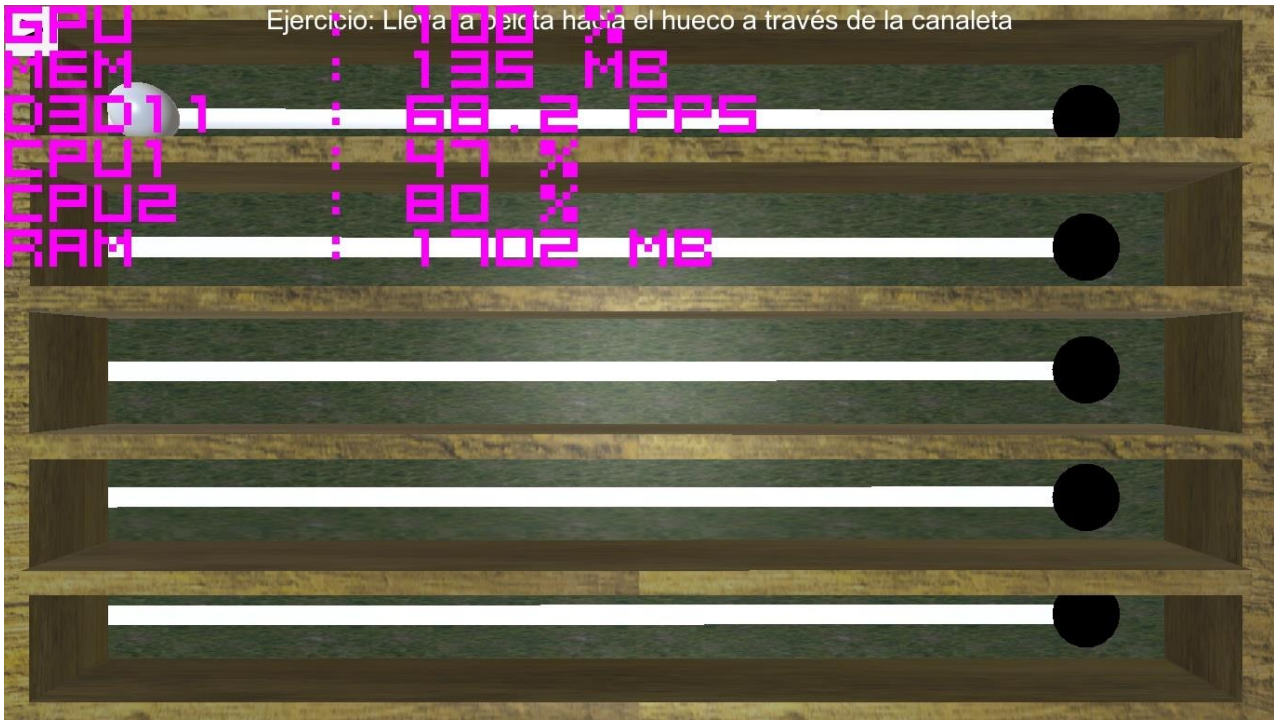


Figura 9.2.13 *Framerate* del programa de minigolf en modo *Simple* ejecutándose en el equipo portátil

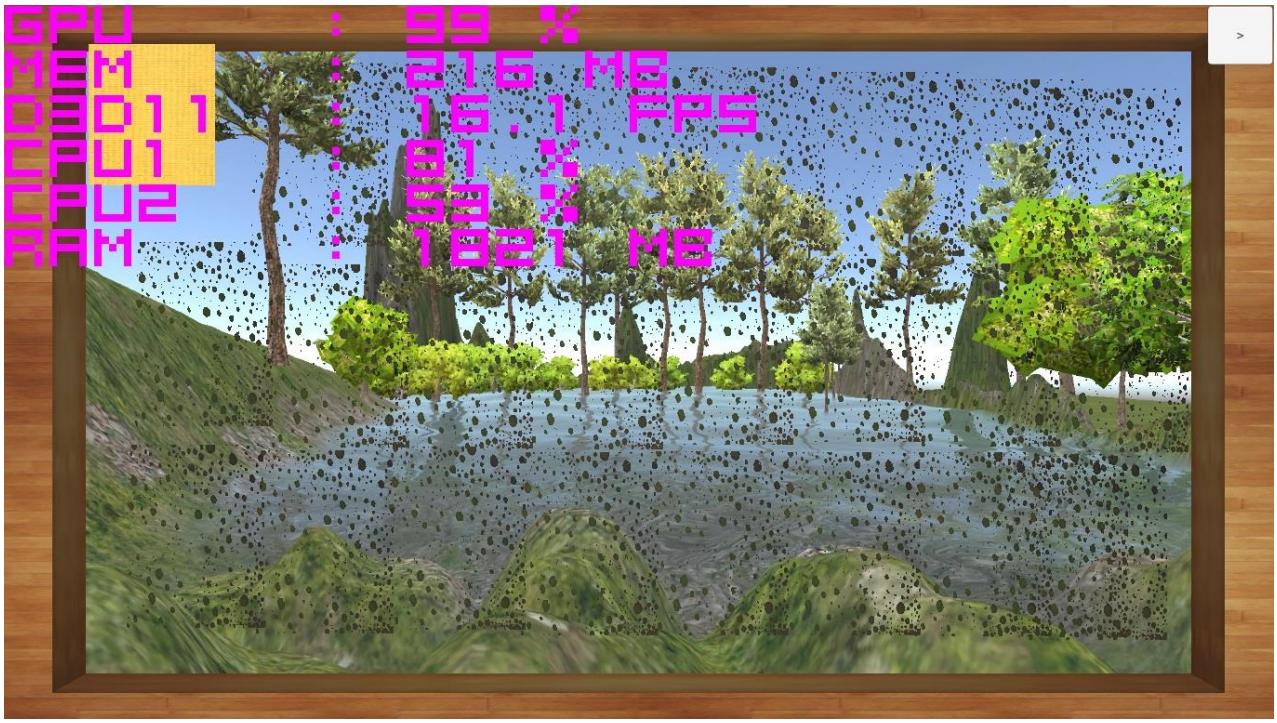


Figura 9.2.14 *Framerate* del programa de limpiar la ventana en modo *Simple* ejecutándose en el equipo portátil

Tal como se había propuesto en un principio y con base en lo que se observó en la tabla, la solución para computadoras con *hardware* limitado, será utilizar un modo de calidad acorde al equipo que se tiene, de acuerdo a la potencia que tenga la tarjeta gráfica del equipo.

Si bien, el programa hace un uso de bajo a moderado de la CPU del sistema, principalmente utiliza la aceleradora gráfica a su máxima capacidad. Una segunda solución aunque muy poco recomendable, sería disminuir la resolución utilizada para intentar mejorar el *framerate*, sin embargo esto no es aconsejable dado que cualquier dispositivo de despliegue moderno basado en LCD, LED o plasma, siempre mostrará imágenes muy nítidas únicamente si estas están en una resolución mayor o igual a su resolución nativa, y el disminuirla causará que la calidad de imagen sea inadecuada o desagradable para el usuario.

9.3 Cambios realizados tras la realización de las pruebas

Con base en los datos obtenidos se realizaron las siguientes modificaciones al penúltimo entregable de los programas:

Programas de minigolf:

*Para todos los modos

- Para la versión de pantalla táctil, se mejoró el algoritmo para el manejo de los puntos de toque sobre la pantalla táctil, de forma que ahora detectará de forma adecuada cuando un paciente colocaba su brazo completo sobre la pantalla intentando mover la pelota o paño con un dedo.

-Se optimizaron los efectos visuales de todos los programas para incrementar el rendimiento, en especial en sistemas con *hardware* limitado.

-Se mejoró el algoritmo para el movimiento de la pelota, con el fin de evitar que la misma quede atascada en alguna de las canaletas cuando se realicen movimientos muy rápidos

Programas de limpiar la ventana:

*Para todos los modos

-Se optimizaron los efectos visuales de todos los programas para incrementar el rendimiento, en especial en sistemas con *hardware* limitado. Se eliminó el movimiento de la textura de las manchas sobre la ventana

-Ahora, cuando se introduce un número mayor a 25 en el apartado de repeticiones, el número se convierte internamente a un máximo de 25, ya que este número indica en realidad la cantidad de repeticiones con que aparecerá la textura de la suciedad en la ventana, pero no se elimina el número como ocurría previamente

9.4 Perspectivas

La terapeuta especialista en Neurorehabilitación, quién utilizó la versión preliminar de los programas, concordó en que este conjunto de ejercicios podrían ser de gran utilidad para las terapias de neurorehabilitación tras un ictus en pacientes que lo requieran. Ya que al ser programas gráficos, resultan más adecuados para los pacientes que realizar ejercicios sin sentido, tales como repeticiones sin un fin determinado de cierto movimiento con el brazo.

Explicó, que esto es debido a que la Neurorehabilitación funcional ha demostrado ser más efectiva en cualquier paciente por el simple hecho de que al paciente se le prepara para actuar en la vida real al tratarlo con ejercicios relacionados con actividades reales. Especialmente, el programa de limpiar la ventana, comentó, un paciente pensaría “Ahora tengo que limpiar esta ventana, entonces lo haré cómo lo aprendí en el ejercicio de computadora de la neurorehabilitación” y al ser tan similar al programa gráfico, el paciente no tendría muchos problemas repitiendo los movimientos horizontales, verticales o circulares que aprendió utilizando los programas. En el caso del programa de minigolf, una opción adicional sería indicar al paciente que intente realizar el ejercicio utilizando únicamente un dedo en lugar de toda la mano, con el fin de ejercitar la coordinación motriz fina. Por supuesto el programa es capaz de funcionar si se utiliza también toda la mano del paciente.

Entonces tal como se planteó desde un principio en los objetivos y en opinión de una terapeuta especialista en Neurorehabilitación, estos programas podrán ser utilizados como una herramienta adicional a las terapias de Neurorehabilitación tradicionales.

9.5 Conclusión

La tecnología es una rama del conocimiento humano que se encuentra en constante evolución, para bien o para mal. Y en muchas ocasiones se considera como algo negativo, principalmente por el hecho de que reduce la dificultad para realizar las diferentes actividades de la vida cotidiana en comparación con la forma en que se hacía en finales del siglo pasado y comienzos del siglo actual. Sin embargo, como cualquier otro conocimiento humano, el problema yace en el uso que se le da y no en el conocimiento en sí mismo.

Para el caso de los programas gráficos y específicamente los que reconocemos como videojuegos, durante muchos años ha existido un prejuicio que afirma que dichos programas solo son videojuegos que poco aportan, y que dañan la vista a las personas.

Sin embargo, en el presente desarrollo de tesis, se presentó un caso en que esta tecnología se podría utilizar para beneficio de un sector particular de la población, específicamente, aquel sector que sufre de afección del movimiento voluntario de la extremidad superior tras un accidente cerebrovascular. Este desarrollo permitirá a un terapeuta, utilizar una herramienta adicional para las terapias de neurorehabilitación que no solo consista de repeticiones sin utilidad, especialmente en el programa de limpiar la ventana, donde podrá practicar diferentes movimientos de la mano, y posiblemente aplicarlos en la vida real al utilizar los movimientos aprendidos en los diferentes modos de uso y dificultad de los programas.

Los eventos cerebrovasculares han existido y seguirán existiendo durante muchos años en la población mundial, y puesto que no siempre es posible recuperar el cien por ciento de la movilidad normal, las terapias de neurorehabilitación tienen por objetivo mejorar la calidad de vida del paciente tanto como sea posible.

Este caso no es único, y tal como se pudo observar en el apartado de “Aplicaciones”, existen ya un sinfín de programas gráficos tanto para computadoras de escritorio como teléfonos móviles, que tienen fines educativos, de investigación, etc. Más allá de los fines de entretenimiento.

Se cumplieron los objetivos establecidos ya que:

-Se desarrolló un conjunto de herramientas de *software* gráficas que servirán como complementos a las terapias convencionales de neurorehabilitación, presentando ejercicios adecuados para neurorehabilitación a modo de juegos que representarán actividades comunes o cotidianas de la vida real dentro de un entorno virtual de computadora

-Se realizaron las adaptaciones necesarias para que las herramientas de *software* utilicen como dispositivos de entrada, en un primer caso el sensor *Kinect One*, y en el segundo caso una pantalla táctil. Las herramientas de *software* entonces responden a las entradas sobre alguno de los dispositivos, en el caso del programa de minigolf, con el fin de mover la pelota hasta su respectivo hueco, o bien en el caso programa de limpiar la ventana, con el fin de mover el paño para limpiar las manchas de la ventana

-Se realizaron diferentes pruebas con las versiones preliminares de las herramientas de *software*, en personas sanas, para conocer las mejoras que pudiera requerir el sistema. Así mismo, se realizaron diferentes pruebas de rendimiento del *software*, por su condición particular de ser programas gráficos con el fin de conocer las optimizaciones que pudieran requerir para funcionar no solo en dispositivos con *hardware* robusto sino también en computadoras con *hardware* limitado.

-Se obtuvieron las versiones finales de los programas, con base a los resultados obtenidos en las pruebas sobre la versión preliminar de los programas. Y se realizaron las optimizaciones necesarias para que el *software* funcione en computadoras con *hardware* limitado.

Apéndices

En los siguientes apéndices, se muestran los *scripts* correspondiendes a cada programa ejecutable respectivo tal como se indica en el apartado de Diagramas del sistema. Considérese que no se muestran *scripts* que no hayan sido desarrollados o modificados por el autor del presente desarrollo de Tesis. Dichos *scripts* generalmente se encargan de otras tareas de manera automática, tal como se explicó en el apartado de Elección de herramientas de *software*.

Apéndice 1

A continuación se muestran los *scripts* correspondientes al programa de minigolf para pantalla táctil.

-log.cs: este *script* contiene el método para ingresar al menú del programa, y contiene el método que permite salir del programa.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class log : MonoBehaviour
{
	public InputField nombre_v;
	string nombre_usuario;
	// Use this for initialization
	void Start ()
	{

	}

	// Update is called once per frame
	void Update ()
	{
		if (Input.GetKeyDown (KeyCode.KeypadEnter) || Input.GetAxis("Submit")>0.0f)
		{Debug.Log ("enter");
			log_in();
		}
	}
	public void log_in()//sirve para ingresar al juego
	{
		if (nombre_v.text != "")//si no se ingresa nada no se puede acceder
		{

			Application.LoadLevel(1);
		}
		else
		{
			Debug.Log ("No se ingreso nada");
		}
	}
	public void salir()//para salir del programa con boton salir
	{
		Application.Quit ();
	}
}
```

- menu.cs: este *script* contiene los métodos necesarios para acceder a alguno de los modos del programa, ya sea en dificultad con o sin canaletas, e ingresar el número de repeticiones deseadas. También contiene métodos para salir del programa o bien volver a la pantalla de *log in*.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class menu : MonoBehaviour
{
	public InputField repeticiones_;
	bool ej1_espejo,ej2_espejo,ej3_espejo,ej1_can,ej2_can,ej3_can;
	public Toggle check1,check2,check3,cs1,cs2,cs3;
	public Image ej1_,ej2_,ej3_;
	public int repeticiones;
	public int orientacion;
	public Sprite ej1m,ej2m,ej3m,ej1_esp,ej2_esp,ej3_esp;
	public Object vacio;
	// Use this for initialization
	void Start ()
	{
		ej1_espejo = false;
		ej2_espejo = false;
		ej3_espejo = false;
		ej1_can = false;//si esta en falso no lleva canaleta, en true lleva canaleta
		ej2_can = false;
		ej3_can = false;
	}
	void Awake()//indicamos que no se destruya vacio, el cual contiene un script con la variable del
	{//numero de repeticiones ingresadas
		DontDestroyOnLoad (vacio);
	}
	// Update is called once per frame
	void Update ()
	{
		if (repeticiones_.text != "")//solo se intenta pasar a enter si hay algo escrito
		{
			System.Int32.TryParse (repeticiones_.text, out repeticiones);
			if(repeticiones==0)//si se ingresa algo que no sea entero, se limpia el panel y no se puede ingresar
			{
				repeticiones_.text="";
			}
		}

	}
	public void salir_()
	{
		Application.Quit ();
	}
	public void regresar()//al regresar a la pantalla de log in, eliminamos vacio para evitar duplicados
	{
		Destroy (vacio);
```

```
		Application.LoadLevel (0);
}
public void juego1()//vertical
{
		if(repeticiones!=0)
		{
				if(ej1_can==false)
			{
				orientacion=1;
				Application.LoadLevel (2);//con canaleta
			}
				else
			{
				orientacion=5;
				Application.LoadLevel (5);//sin canaleta
			}
		}
}
public void juego2()//horizontal
{
		if (repeticiones != 0)
		{
			if (ej2_can == false)
			{   orientacion=0;
				Application.LoadLevel (4);//con canaleta
			}
			else
			{   orientacion=3;
				Application.LoadLevel (3);//sin canaleta
			}
		}
}
public void juego3()//combinado
{
		if (repeticiones != 0)
		{
			if (ej3_can == false)
			{
				orientacion=2;
				Application.LoadLevel (6);//con canaleta
			}
			else
			{
				orientacion=4;
				Application.LoadLevel (7);//sin canaleta
			}
		}
}
public void check_1()
{
		ej1_espejo = check1.isOn;
		if (ej1_espejo == false)
			ej1_.sprite = ej1m;
		else
			ej1_.sprite = ej1_esp;
}
public void check_2()
{
		ej2_espejo = check2.isOn;
		if (ej2_espejo == false)
			ej2_.sprite = ej2m;
		else
			ej2_.sprite = ej2_esp;
}
public void check_3()
{
		ej3_espejo = check3.isOn;
		if (ej3_espejo == false)
```

```
                ej3_.sprite = ej3m;
            else
                ej3_.sprite = ej3_esp;
        }
        ///con o sin canaleta
        public void cscan_1()
        {
            ej1_can = cs1.isOn;
            if (ej1_can == false)
                ej2_.sprite = ej2m;
            else
                ej2_.sprite = ej2_esp;
        }
        public void cscan_2()
        {
            ej2_can = cs2.isOn;
            if (ej2_can == false)
                ej1_.sprite = ej1m;
            else
                ej1_.sprite = ej1_esp;
        }
        public void cscan_3()
        {
            ej3_can = cs3.isOn;
            if (ej3_can == false)
                ej3_.sprite = ej3m;
            else
                ej3_.sprite = ej3_esp;
        }
}
```

- pelota_beh.cs: Este *script* contiene métodos que se encargan tanto del movimiento de la pelota, como de realizar todas las comprobaciones necesarias para el funcionamiento del programa, en cualquiera de los modos y en cualquiera de las dos dificultades (con o sin canaleta). Adicionalmente, contiene métodos para salir del programa, reiniciar la partida actual o bien regresar el menú de configuración.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System.Collections.Generic;
using UnityStandardAssets.CrossPlatformInput;
public class pelota_beh : MonoBehaviour
{
    Rigidbody pelota_rig;
    Vector3 mouse_mov,pelota_pos,pos_ini;
    Vector3[] pos_cam=new Vector3[2]{new Vector3(0.0f,11.76f,-3.5f),new Vector3(0.0f,11.76f,3.5f)};
    Vector3[] pos_camh=new Vector3[2]{new Vector3(0.0f,14.3f,-4.54f),new Vector3(0.0f,14.3f,6.77f)};
    public GameObject pelotaa,piso;
    public Camera cam;
    Ray rayo;
    int choques,choques_aux,orien;
    int repe,can=0,primeravez=0;
    int repepp=0;
    RaycastHit lugar;
    public Text mensajes;
    SphereCollider esfer;
    GameObject vacio_;
    public GameObject menub,reinib,salirb;
    MeshRenderer pel;
    int[] errores;
```

```
	Vector3[] pos_pelota=new Vector3[5]{new Vector3(-4.16f,1.467f,-7.689f),new Vector3(-4.16f,1.467f,-3.9f),new Vector3(-4.16f,1.467f,-0.04f),new Vector3(-4.16f,1.467f,3.9f),
		new Vector3(-4.16f,1.467f,7.67f)};
	Vector3[] pos_pelotah=new Vector3[5]{new Vector3(14.647f,3.683f,-7.37f),new Vector3(14.647f,3.683f,-3.61f),new Vector3(14.647f,3.683f,0.24f),new Vector3(14.647f,3.683f,4.14f),
		new Vector3(14.647f,3.683f,7.71f)};
	Vector3[] pos_pelotah_sc=new Vector3[5]{new Vector3(-3.27f,1.766f,-6.23f),new Vector3(-1.559f,1.766f,-6.23f),new Vector3(0.12f,1.766f,-6.23f),new Vector3(1.826f,1.766f,-6.23f),
		new Vector3(3.423f,1.766f,-6.23f)};
	Vector3[] pos_pelotav_sc=new Vector3[5]{new Vector3(6.15f,1.2f,-3.57f),new Vector3(3.06f,1.2f,-3.57f),new Vector3(-0.02f,1.2f,-3.57f),new Vector3(-3.06f,1.2f,-3.57f),
		new Vector3(-6.05f,1.2f,-3.57f)};
	Vector3 pos_pelotacom = new Vector3 (4.897f, 0.405f, -2.38f);
	Vector3 pos_pelotacom_sc = new Vector3 (3.313f, 0.602f, 6.751f);
	bool acabo=false,menuon=false;
	// Use this for initialization
	void Start ()
	{
		pel = pelotaa.GetComponent<MeshRenderer> ();
		vacio_ = GameObject.Find ("vacio");
		pos_ini=pelotaa.transform.position;
		pelota_rig = GetComponent<Rigidbody> ();
		esfer = GetComponent<SphereCollider> ();
		repe = vacio_.GetComponent<menu> ().repeticiones;
		orien = vacio_.GetComponent<menu> ().orientacion;Debug.Log (orien);
		errores=new int[repe];
		limpia_e ();
		menub.SetActive (false);
		reinib.SetActive (false);
		salirb.SetActive (false);
		//if(orien==1)
		//	cam.transform.position = pos_cam [0];
		//if(orien==0)
		//	cam.transform.position = pos_camh [0];
		repetir();
	}
	// Update is called once per frame
	void Update ()
	{
		if (transform.position.y < -3.0f)
		{
			if(repepp<repe)
			  repetir();
			else
			{
				mensajes.text="Ejercicio terminado\nTuviste: "+choques+" errores totales\n";
				for(int j=0;j<repe;j++)
				{
					mensajes.text+="Repeticion "+(j+1)+": "+errores[j]+" errores\n";
				}
				pelotaa.SetActive(false);
				can=0;
			}
		}
		On ();
		//pelota_rig.AddForce (new Vector3 (0.0f, 1000.0f, 0.0f));

	}
	void repetir()
	{
		pel.enabled = false;
		//mensajes.text = "Repeticion " + (repepp + 1);
		//StartCoroutine ("posiciona_cam");
		pel.enabled = true;
		reinicio ();
		mensajes.text = "";
		//if(acabo)
		//	StopCoroutine ("repetir");
```

```
	}
	IEnumerator posiciona_cam()
	{
		if(primeravez!=0)
		{
			if (can == 3)
			{
				acabo=true;
			}
			if (can == 0)
			{
				acabo=true;
			}
			yield return new WaitForSeconds (0.5f);
		}
	}
	 void On()
	{
		//////////////////////
		float distancia_t=0.0f;
		int toques = gameObject.GetComponent<TuioInput> ().num_toques;
		Vector2[] toques_pos = new Vector2[2];
		if (toques == 1)
		{
			toques_pos[0]=gameObject.GetComponent<TuioInput> ().toque12[0];
			rayo = cam.ScreenPointToRay (new Vector3(toques_pos[0].x,toques_pos[0].y,0.0f));
		}
		if (toques == 2)
		{
			Vector2 promedio;
			toques_pos[0]=gameObject.GetComponent<TuioInput> ().toque12[0];
			toques_pos[1]=gameObject.GetComponent<TuioInput> ().toque12[1];
			distancia_t=Vector2.Distance(toques_pos[0],toques_pos[1]);
			float	d_toque1_pelota=Vector3.Distance(transform.position,Camera.main.ScreenToWorldPoint(new
Vector3(toques_pos[0].x,toques_pos[0].y,7.67f)));
			float	d_toque2_pelota=Vector3.Distance(transform.position,Camera.main.ScreenToWorldPoint(new
Vector3(toques_pos[1].x,toques_pos[1].y,7.67f)));
			if(distancia_t<120.0f)
			{
				promedio=(toques_pos[0]+toques_pos[1])/2;
				rayo = cam.ScreenPointToRay (new Vector3(promedio.x,promedio.y,0.0f));
			}
			else
			{
				if(d_toque1_pelota<d_toque2_pelota)
					rayo		=		cam.ScreenPointToRay		(new
Vector3(toques_pos[0].x,toques_pos[0].y,0.0f));
				if(d_toque2_pelota<d_toque1_pelota)
					rayo		=		cam.ScreenPointToRay		(new
Vector3(toques_pos[1].x,toques_pos[1].y,0.0f));
			}
		}
		if (toques == 1 || (toques == 2))
		{
			Physics.Raycast (rayo, out lugar);
			if (lugar.collider.name == "pelota")
			{
				pelota_pos = lugar.point;
				pelota_pos.y = pos_ini.y;//0.4599102f;
				pelotaa.transform.position = pelota_pos;
			}
		}
	}
	public void salir()
	{
		Application.Quit ();
	}
```

```
	public void reinicio()
	{
		acabo = false;
		pelotaa.SetActive (false);
		esfer.enabled=true;
		mensajes.text=" ";
		//choques = 0;
		pelota_rig.isKinematic = true;
		//pelota_rig.isKinematic = false;
		pelotaa.gameObject.SetActive (true);
		if(orien==1)
			pelotaa.transform.position = pos_pelota[can];//vertical
		if(orien==0)
			pelotaa.transform.position = pos_pelotah[can];//horizontal
		if(orien==2)
			pelotaa.transform.position = pos_pelotacom;//combinado
		if(orien==3)
			pelotaa.transform.position = pos_pelotah_sc[can];//horizontal sin canaleta
		if(orien==5)
			pelotaa.transform.position = pos_pelotav_sc[can];//vertical sin canaleta
		if(orien==4)
			pelotaa.transform.position = pos_pelotacom_sc;//combinado sin canaleta
		Renderer aux;
		aux=GetComponent<Renderer>();
		aux.material.color=new Color(1.0f,1.0f,1.0f);
	}
	public void reini_manual()
	{
		acabo = false;
		limpia_e ();
		repepp = 0;
		choques = 0;
		can = 0;
		primeravez = 0;
		/*if(orien==1)
			cam.transform.position = pos_cam [0];
		if(orien==0)
			cam.transform.position = pos_camh [0];*/
		reinicio ();
		repetir();
		despliega ();
	}
	public void menu()
	{
		Destroy (vacio_);
		Application.LoadLevel (1);
	}
	public void despliega()//muestra los botones de reinicio salir y menu
	{
		menuon = !menuon;
		menub.SetActive (menuon);
		salirb.SetActive (menuon);
		reinib.SetActive (menuon);
	}
	void OnTriggerEnter(Collider conque)
	{
		if (conque.name == "can1" || conque.name == "can2"||conque.name == "can3"||conque.name == "can4"||conque.name == "can5"||conque.name=="can6"||

conque.name=="can7"||conque.name=="can8"||conque.name=="can9"||conque.name=="can10"||conque.name=="can11"||conque.name=="can12"||conque.name=="can13"||conque.name=="can14"
		    ||conque.name=="can15"||conque.name=="can0")
		{
			choques++;
			errores[repepp]+=1;
			Renderer aux;
			aux=GetComponent<Renderer>();
			aux.material.color=new Color(1.0f,0.0f,0.0f);
```

```
                        pelota_rig.isKinematic =false;
                }
                if (conque.name == "Cube"||conque.name == "Cube2"||conque.name == "Cube3"||conque.name ==
"Cube4"||conque.name == "Cube5")
                {
                        repepp+=1;
                        Debug.Log ("caer");
                        pelota_rig.isKinematic =false;
                        pelota_rig.AddForce(new Vector3(0.5f,0.0f,0.5f));
                        esfer.enabled=false;
                        if(can<4)
                                can++;
                        else
                                can=0;
                        primeravez=1;
                }
        }
        void OnTriggerStay(Collider conque)
        {
                pelota_rig.AddForce(new Vector3(0.5f,0.0f,0.5f));
        }
        void OnTriggerExit(Collider conque)
        {
                Renderer aux;
                aux=GetComponent<Renderer>();
                aux.material.color=new Color(1.0f,1.0f,1.0f);
                pelota_rig.isKinematic =true;
        }
        void limpia_e()//pone en ceros el arreglo de errores
        {
                for (int i=0; i<repe; i++)
                {
                        errores[i]=0;
                }
        }
}
```

- TuioInput.cs: Este *script* forma parte del *plugin* mencionado en el apartado de “Proceso de desarrollo de los programas”. Se encarga de la detección de toques sobre cualquier dispositivo compatible. Únicamente fue modificado para comunicarse con el *script* pelota_beh.cs y enviarle la posición actual de los dos toques detectados sobre la pantalla (en caso de existir).

You should have received a copy of the GNU Lesser Public License
along with Unity3d-TUIO. If not, see <http://www.gnu.org/licenses/>.

If you have any questions regarding this library, or would like to purchase
a commercial licence, please contact Mindstorm via www.mindstorm.com.
*/

```
using System.Collections;
using UnityEngine;
using System.Linq;

/// <summary>
/// Provides TUIO input as UnityEngine.Touch objects for receiving touch information
/// Must be attached to a GameObject in the Hierachy to be used.
///
/// Provides exactly the same interface as UnityEngine.Input regarding touch data
/// allowing any code using UnityEngine.Input to use TuioInput instead.
/// </summary>
public class TuioInput : MonoBehaviour
{
	static TuioComponentBase tracking;

	static Touch[] frameTouches = new Touch[0];

	public static readonly bool multiTouchEnabled = true;
	public Vector2[] toque12 = new Vector2[2];
	public int num_toques = 0;
	public static int touchCount
	{
		get;
		private set;
	}

	void Awake()
	{
		tracking = InitTracking(new TuioTrackingComponent());
	}

	void Update()
	{
		if (tracking == null)
		{
			enabled = false;
			return;
		}
		TuioComponentBase tr = tracking;
		UpdateTouches(tr);
	}

	void UpdateTouches(TuioComponentBase tr)
	{
		tr.BuildTouchDictionary();
		frameTouches = tr.AllTouches.Values.Select(t => t.ToUnityTouch()).ToArray();
		touchCount = frameTouches.Length;
		num_toques = touchCount;
		if (num_toques == 1)
		{
			toque12 [0] = frameTouches [0].position;
		}
		if (num_toques == 2)
		{
			toque12 [0] = frameTouches [0].position;
			toque12 [1] = frameTouches [1].position;
		}
	}

	TuioComponentBase InitTracking(TuioComponentBase tr)
	{
```

```
                tr.ScreenWidth = Camera.main.pixelWidth;
                //tr.ScreenWidth = Screen.width;
                tr.ScreenHeight = Camera.main.pixelHeight;
                //tr.ScreenHeight = Screen.height;
                return tr;
        }

        public static Touch GetTouch(int index)
        {
                return frameTouches[index];
        }

        public static Touch[] touches
        {
                get
                {
                        return frameTouches;
                }
        }

        void OnApplicationQuit()
        {
                if (tracking != null) tracking.Close();
        }
}
```

Apéndice 2

A continuación se muestran los *scripts* correspondientes al programa de limpiar la ventana para pantalla táctil.

-log.cs: este *script* contiene el método para ingresar al menú del programa, y contiene el método que permite salir del programa.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class log : MonoBehaviour
{
	public InputField nombre_v;
	string nombre_usuario;
	// Use this for initialization
	void Start ()
	{

	}

	// Update is called once per frame
	void Update ()
	{
		if (Input.GetKeyDown (KeyCode.KeypadEnter) || Input.GetAxis("Submit")>0.0f)
		{Debug.Log ("enter");
			log_in();
		}
	}
	public void log_in()//sirve para ingresar al juego
	{
		if (nombre_v.text != "")//si no se ingresa nada no se puede acceder
		{

			Application.LoadLevel(1);
		}
		else
		{
			Debug.Log ("No se ingreso nada");
		}
	}
	public void salir()//para salir del programa con boton salir
	{
		Application.Quit ();
	}
}
```

- menu_beh_.cs: este *script* contiene los métodos que permiten ingresar a cualquiera de los tres modos del programa e indicar el número de repeticiones deseada (el número de manchas de suciedad que habrá en la ventana para limpiarlas). Adicionalmente contiene métodos para salir del programa o bien regresar a la pantalla de *log in*.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
public class menu_beh_ : MonoBehaviour
{
	public Toggle check;
	public InputField repeticions_;
	//bool espejo;
```

```
	public int repe,modo;//los modos son: 0-horizontal 1-vertical 2-libre
	// Use this for initialization
	void Start ()
	{
		//espejo = false;

	}
	void Awake()//indicamos que no se destruya vacio, el cual contiene un script con la variable del
	{//numero de repeticiones ingresadas
		DontDestroyOnLoad (this);
	}
	// Update is called once per frame
	void Update ()
	{
		if (repeticions_.text != "")//solo se intenta pasar a enter si hay algo escrito
		{
			System.Int32.TryParse (repeticions_.text, out repe);
			if(repe==0)//si se ingresa algo que no sea entero, se limpia el panel y no se puede ingresar
			{
				repeticions_.text="";
			}
			if(repe>25)
			{
				repe=25;
			}
		}
	}
	public void regresa()
	{
		Destroy (this);
		Application.LoadLevel (0);
	}
	public void sal()
	{
		Application.Quit ();
	}
   public void horiz()//modo horizontal
	{
		if (repe > 0&&repe<=25)
		{
			modo = 0;
			Application.LoadLevel (2);
		}
	}
	public void verti()//modo vertical
	{
		if (repe > 0&&repe<=25)
		{
			modo = 1;
			Application.LoadLevel (2);
		}
	}
	public void libre()//modo en circulos
	{
		if (repe > 0&&repe<=25)
		{
				modo=2;
				Application.LoadLevel (2);
		}
	}
}
```

- mov_limpiador: Este *script* contiene los métodos necesarios para el movimiento de la esponja para limpiar, así como de todas las comprobaciones necesarias para el funcionamiento del programa. Contiene también un método para reiniciar la partida actual.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

//////////////////////
////
///
///

public class mov_limpiador : MonoBehaviour
{
	TuioInput touch_man;
	Ray rayo,rayoaux;
	public Camera cam;
	RaycastHit lugar,lugaraux;
	Vector3 lim_pos;
	Vector3[] cache=new Vector3[2];
//	Vector3 resul;
	float y_ini;
	LineRenderer rastro;
	int semaforo=0,indice=0,contador=0;
	GameObject menu;
	public Text direccion,Debugeer;
//	Object aux_o1,aux_o2;
	int repeticiones,posicion;
	string[] dir=new string[2];
	bool[] esp_t=new bool[3];//indicadores de acabar de limpiar la barra de espuma
	bool[] det=new bool[25];
	bool corretiempo,terminado=false,flag=false;
	public GameObject[] manchas=new GameObject[15];
	float tiempo=0.0f;
	public Text cronometro;
	public GameObject reini_b,salir_b,regres_b;
	Renderer[] espuma_rend=new Renderer[25]{null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null};
	Vector3[] posmugre_hor=new Vector3[25]{new Vector3(0.453f,0.25f,0.696f),new Vector3(0.451f,0.25f,0.517f),new Vector3(0.449f,0.25f,0.279f),new Vector3(0.44f,0.25f,0.0f),new Vector3(0.443f,0.25f,-0.256f),
		new Vector3(0.431f,0.25f,-0.559f),new Vector3(0.416f,0.25f,-0.83f),new Vector3(0.395f,0.25f,-1.032f),new Vector3(0.061f,0.25f,0.948f),new Vector3(0.013f,0.25f,0.65f),
		new Vector3(0.012f,0.25f,0.38f),new Vector3(0.01f,0.25f,0.098f),new Vector3(0.019f,0.25f,-0.187f),new Vector3(0.025f,0.25f,-0.473f),new Vector3(0.015f,0.25f,-0.766f),
		new Vector3(0.003f,0.25f,-1.046f),new Vector3(-0.38f,0.25f,0.966f),new Vector3(-0.392f,0.25f,0.685f),new Vector3(-0.384f,0.25f,0.397f),new Vector3(-0.396f,0.25f,0.108f),
		new Vector3(-0.395f,0.25f,-0.177f),new Vector3(-0.396f,0.25f,-0.464f),new Vector3(-0.397f,0.25f,-0.75f),new Vector3(-0.4f,0.25f,-0.916f),new Vector3(-0.405f,0.25f,-1.0f)};

	Vector3[] posmugre_cir=new Vector3[25]{new Vector3(0.486f,0.25f,0.57f),new Vector3(0.205f,0.25f,0.696f),new Vector3(-0.1f,0.25f,0.914f),new Vector3(-0.402f,0.25f,0.8f),new Vector3(-0.444f,0.25f,0.407f),
		new Vector3(-0.475f,0.25f,0.015f),new Vector3(-0.487f,0.25f,-0.376f),new Vector3(-0.326f,0.25f,-0.775f),new Vector3(-0.037f,0.25f,-0.908f),new Vector3(0.275f,0.25f,-0.905f),
		new Vector3(0.47f,0.25f,-0.548f),new Vector3(0.47f,0.25f,-0.15f),new Vector3(0.243f,0.25f,0.088f),new Vector3(-0.027f,0.25f,0.032f),new Vector3(0.06f,0.25f,-0.365f),
		new Vector3(0.488f,0.25f,-0.321f),new Vector3(0.212f,0.25f,-0.203f),new Vector3(-0.055f,0.25f,-0.324f),new Vector3(-0.307f,0.25f,-0.316f),new Vector3(-0.489f,0.25f,-0.314f),
		new Vector3(0.459f,0.25f,-0.915f),new Vector3(0.211f,0.25f,-0.608f),new Vector3(-0.046f,0.25f,-0.852f),new Vector3(-0.287f,0.25f,-0.884f),new Vector3(-0.451f,0.25f,-0.625f)};
```

```
float till=0.0f;
// Use this for initialization
void Start ()
{
	multitoque_init ();//inicializar funciones multitoque

	direccion.text = "";
	dir [0] = "";
	dir [1] = "";
	for (int i=0; i<25; i++)
	{
		det[i]=false;
	}
	for (int i=0; i<3; i++)
	{
		esp_t[i]=false;
	}
	menu = GameObject.Find ("menu_beh");
	rastro = GetComponent<LineRenderer> ();
	y_ini = transform.position.y;
	repeticiones = menu.GetComponent<menu_beh_> ().repe;
	posicion = menu.GetComponent<menu_beh_> ().modo;
	//Debug.Log (repeticiones);
	//Debug.Log (posicion);
	for (int i=0; i<25; i++)
	{
		manchas[i].SetActive(false);
	}
	creaespuma ();
	for (int i=0; i<25; i++)
	{
		espuma_rend[i]=manchas[i].GetComponent<Renderer>();
	}
}

// Update is called once per frame
void Update ()
{//Debug.Log (Input.mousePosition);
	bool cache = false;
	if (corretiempo)
		tiempo += Time.deltaTime;
	//Debug.Log (tiempo);
	//Input.multiTouchEnabled = true;

	for (int i=0; i<repeticiones; i++)//si ya se quito toda la mugre se termina el juego
	{
		if(i==0)
			terminado=det[0];
		else
			terminado=terminado&&det[i];
	}
	if (terminado)//comprobacion de terminacion
	{
		corretiempo=false;
		Debug.Log ("termino");
		float tiempo_truncado=Mathf.Round(tiempo*100.0f)/100.0f;
		cronometro.text="Tiempo total : "+tiempo_truncado+" segundos ";
		gameObject.SetActive(false);
	}
	////movimiento de la textura de la mugre
	if (till < 0.05f)
		till += 0.0001f * Time.deltaTime;
	if (posicion == 0)//modo horizontal
	{
		for (int i=0; i<25; i++)
		{
			espuma_rend [i].material.mainTextureOffset = new Vector2 (0.0f, till);
		}
```

```
		}
		if (posicion == 1)//modo vertical
		{
			for (int i=0; i<25; i++)
			{
				espuma_rend [i].material.mainTextureOffset = new Vector2 (till, 0.0f);
			}
		}
		On ();
	}
	void On()//originalmente onmouseover
	{
		//pelota_rig.isKinematic = false;
		//seguir = true;
		//if(seguir)
		//{
		float distancia_t=0.0f;
		int toques = gameObject.GetComponent<TuioInput> ().num_toques;
		Vector2[] toques_pos = new Vector2[2];
		if (toques == 1)
		{//Debug.Log ("un toque");
			toques_pos[0]=gameObject.GetComponent<TuioInput> ().toque12[0];
			rayo = cam.ScreenPointToRay (new Vector3(toques_pos[0].x,toques_pos[0].y,0.0f));
		}
		if (toques == 2)
		{
			Vector2 promedio;
			toques_pos[0]=gameObject.GetComponent<TuioInput> ().toque12[0];
			toques_pos[1]=gameObject.GetComponent<TuioInput> ().toque12[1];
			distancia_t=Vector2.Distance(toques_pos[0],toques_pos[1]);
			float	d_toque1_pelota=Vector3.Distance(transform.position,Camera.main.ScreenToWorldPoint(new
Vector3(toques_pos[0].x,toques_pos[0].y,1.038f)));
			float	d_toque2_pelota=Vector3.Distance(transform.position,Camera.main.ScreenToWorldPoint(new
Vector3(toques_pos[1].x,toques_pos[1].y,1.038f)));
			//direccion.text="toque 1 a:"+d_toque1_pelota+"de distancia";
			//direccion.text+="toque 2 a:"+d_toque2_pelota+"de distancia";
			if(distancia_t<120.0f)
			{
			promedio=(toques_pos[0]+toques_pos[1])/2;
			rayo = cam.ScreenPointToRay (new Vector3(promedio.x,promedio.y,0.0f));
			}
			else
			{
				if(d_toque1_pelota<d_toque2_pelota)
					rayo		=		cam.ScreenPointToRay		(new
Vector3(toques_pos[0].x,toques_pos[0].y,0.0f));
				if(d_toque2_pelota<d_toque1_pelota)
					rayo		=		cam.ScreenPointToRay		(new
Vector3(toques_pos[1].x,toques_pos[1].y,0.0f));

			}
		}
		if (toques==1||(toques==2))
	{
			Physics.Raycast (rayo, out lugar);Debugeer.text=lugar.collider.name;//Raycast para definir posicion del
trapo
			if(lugar.collider.name=="limpiador")
		 {
			lim_pos = lugar.point;
			lim_pos.y = y_ini;
			transform.position = lim_pos;
			rayoaux.origin = transform.position;
			rayoaux.direction = new Vector3 (0.0f, -1.0f, 0.0f);
			Physics.Raycast (rayoaux, out lugaraux);//Raycast para detectar si hay manchas de suciedad debajo
del trapo
			//direccion.text=lugaraux.collider.name;//Debug.Log (lugaraux.collider.name);
			if (lugaraux.collider != null && lugaraux.collider.name != "ventana" && lugaraux.collider.name !=
"limpiador")
```

```
                {
                        corretiempo = true;
                        if(contador<repeticiones)
                        {
                                int indice=contador+1;
                                string nombre="espuma"+indice;//direccion.text=nombre;//Debug.Log (nombre);
                                if (lugaraux.collider.name == nombre)
                                {
                                        manchas [contador].SetActive (false);
                                        det [contador] = true;
                                        contador++;
                                }
                        }
                }
         }
        }
}
void creaespuma()
{
        if (posicion == 0)
        {//se eligio modo horizontal
                Quaternion auxi_rot=Quaternion.identity;
                for(int i=0;i<repeticiones;i++)
                {
                        manchas[i].transform.position=posmugre_hor[i];
                        auxi_rot=manchas[i].transform.localRotation;
                        auxi_rot.eulerAngles=new Vector3(90.0f,270.0f,0.0f);
                        manchas[i].transform.localRotation=auxi_rot;
                }
        }
        if (posicion == 1)//se eligio modo vertical
        {
                //se deja tal cual estan las manchas
        }
        if (posicion == 2)
        {//se eligio modo en circulos
                int aux_rep=0;
                if(repeticiones>15)
                {
                        aux_rep=15;
                        repeticiones=aux_rep;
                }
                else
                        aux_rep=repeticiones;

                for(int i=0;i<aux_rep;i++)
                {
                        manchas[i].transform.position=posmugre_cir[i];
                }
                for (int i=0; i<aux_rep; i++)
                {
                        manchas [i].SetActive (true);
                }
        }
        if (posicion == 0 || posicion == 1)
        {
                for (int i=0; i<repeticiones; i++)
                {
                        manchas [i].SetActive (true);
                }
        }
}
public void reinicio()
{
        rastro.SetVertexCount(0);
        corretiempo = false;
        tiempo = 0.0f;
        cronometro.text = "";
```

```
		contador = 0;
		till = 0.0f;
		for (int i=0; i<25; i++)
		{
			det[i]=false;
		}
		for (int i=0; i<3; i++)
		{
			esp_t[i]=false;
		}
		transform.position = new Vector3 (0.549f, 0.28f, 1.038f);
		for (int i=0; i<25; i++)
		{
			manchas[i].SetActive(false);
		}
		creaespuma ();
		gameObject.SetActive(true);
		replie_desplie ();
	}
	void multitoque_init()
	{
		Input.multiTouchEnabled = true;
	}
	public void replie_desplie()//repliega o despliega el menu
	{
		flag = !flag;
		reini_b.SetActive (flag);
		salir_b.SetActive (flag);
		regres_b.SetActive (flag);
	}
}
```

- TuioInput.cs: Este *script* forma parte del *plugin* mencionado en el apartado de “Proceso de desarrollo de los programas”. Se encarga de la detección de toques sobre cualquier dispositivo compatible. Únicamente fue modificado para comunicarse con el *script* mov_limpiador.cs y enviarle la posición actual de los dos toques detectados sobre la pantalla (en caso de existir).

```
/*
Unity3d-TUIO connects touch tracking from a TUIO to objects in Unity3d.

Copyright 2011 - Mindstorm Limited (reg. 05071596)

Author - Simon Lerpiniere

This file is part of Unity3d-TUIO.

Unity3d-TUIO is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

Unity3d-TUIO is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Lesser Public License for more details.

You should have received a copy of the GNU Lesser Public License
along with Unity3d-TUIO.  If not, see <http://www.gnu.org/licenses/>.

If you have any questions regarding this library, or would like to purchase
a commercial licence, please contact Mindstorm via www.mindstorm.com.
*/
```

```
using System.Collections;
using UnityEngine;
using System.Linq;

/// <summary>
/// Provides TUIO input as UnityEngine.Touch objects for receiving touch information
/// Must be attached to a GameObject in the Hierachy to be used.
///
/// Provides exactly the same interface as UnityEngine.Input regarding touch data
/// allowing any code using UnityEngine.Input to use TuioInput instead.
/// </summary>
public class TuioInput : MonoBehaviour
{
	static TuioComponentBase tracking;

	static Touch[] frameTouches = new Touch[0];

	public static readonly bool multiTouchEnabled = true;
	public Vector2[] toque12 = new Vector2[2];
	public int num_toques = 0;
	public static int touchCount
	{
		get;
		private set;
	}

	void Awake()
	{
		tracking = InitTracking(new TuioTrackingComponent());
	}

	void Update()
	{
		if (tracking == null)
		{
			enabled = false;
			return;
		}
		TuioComponentBase tr = tracking;
		UpdateTouches(tr);
	}

	void UpdateTouches(TuioComponentBase tr)
	{
		tr.BuildTouchDictionary();
		frameTouches = tr.AllTouches.Values.Select(t => t.ToUnityTouch()).ToArray();
		touchCount = frameTouches.Length;
		num_toques = touchCount;
		if (num_toques == 1)
		{
			toque12 [0] = frameTouches [0].position;
		}
		if (num_toques == 2)
		{
			toque12 [0] = frameTouches [0].position;
			toque12 [1] = frameTouches [1].position;
		}
	}

	TuioComponentBase InitTracking(TuioComponentBase tr)
	{
		tr.ScreenWidth = Camera.main.pixelWidth;
		//tr.ScreenWidth = Screen.width;
		tr.ScreenHeight = Camera.main.pixelHeight;
		//tr.ScreenHeight = Screen.height;
		return tr;
	}
```

```
	public static Touch GetTouch(int index)
	{
		return frameTouches[index];
	}

	public static Touch[] touches
	{
		get
		{
			return frameTouches;
		}
	}

	void OnApplicationQuit()
	{
		if (tracking != null) tracking.Close();
	}
}
```

- regre_sal.cs: Este *script* únicamente contiene dos métodos, para regresar al menú de configuración y salir del programa respectivamente.

```
using UnityEngine;
using System.Collections;
public class regre_sal : MonoBehaviour
{
	GameObject menuu;
	void Start()
	{
		menuu = GameObject.Find ("menu_beh");
	}
	public void regresar()
	{
		Destroy (menuu);
		Application.LoadLevel (1);
	}
	public void salir()
	{
		Application.Quit ();
	}
}
```

Apéndice 3

A continuación se muestran los *scripts* correspondientes al programa de minigolf para sensor *Kinect*.

- log.cs: Este *script* contiene los métodos necesarios para ingresar al programa, y para salir del mismo.

```
 using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class log : MonoBehaviour
{
	public InputField nombre_v;
	string nombre_usuario;
	// Use this for initialization
	void Start ()
	{

	}

	// Update is called once per frame
	void Update ()
	{
		if (Input.GetKeyDown (KeyCode.KeypadEnter) || Input.GetAxis("Submit")>0.0f)
		{Debug.Log ("enter");
			log_in();
		}
	}
	public void log_in()//sirve para ingresar al juego
	{
		if (nombre_v.text != "")//si no se ingresa nada no se puede acceder
		{

			Application.LoadLevel(1);
		}
		else
		{
			Debug.Log ("No se ingreso nada");
		}
	}
	public void salir()//para salir del programa con boton salir
	{
		Application.Quit ();
	}
}
```

- menu.cs: Este *script* contiene los métodos necesarios para iniciar el programa en cualquiera de los modos posibles y con dificultad con o sin canaleta. Además contiene métodos para salir del programa o bien regresar a la pantalla de *log in*.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class menu : MonoBehaviour
{
	public InputField repeticiones_;
	bool ej1_espejo,ej2_espejo,ej3_espejo,ej1_can,ej2_can,ej3_can;
	public Toggle check1,check2,check3,cs1,cs2,cs3;
	public Image ej1_,ej2_,ej3_;
```

```
public int repeticiones;
public int orientacion;
public Sprite ej1m,ej2m,ej3m,ej1_esp,ej2_esp,ej3_esp;
public Object vacio;
// Use this for initialization
void Start ()
{
	ej1_espejo = false;
	ej2_espejo = false;
	ej3_espejo = false;
	ej1_can = false;//si esta en falso no lleva canaleta, en true lleva canaleta
	ej2_can = false;
	ej3_can = false;
}
void Awake()//indicamos que no se destruya vacio, el cual contiene un script con la variable del
{//numero de repeticiones ingresadas
	DontDestroyOnLoad (vacio);
}
// Update is called once per frame
void Update ()
{
	if (repeticiones_.text != "")//solo se intenta pasar a enter si hay algo escrito
	{
		System.Int32.TryParse (repeticiones_.text, out repeticiones);
		if(repeticiones==0)//si se ingresa algo que no sea entero, se limpia el panel y no se puede ingresar
		{
			repeticiones_.text="";
		}
	}

}
public void salir_()
{
	Application.Quit ();
}
public void regresar()//al regresar a la pantalla de log in, eliminamos vacio para evitar duplicados
{
	Destroy (vacio);
	Application.LoadLevel (0);
}
public void juego1()//vertical
{
	if(repeticiones!=0)
	{
			if(ej1_can==false)
		{
			  orientacion=1;
			  Application.LoadLevel (2);//con canaleta
		}
			else
		{
			  orientacion=5;
			  Application.LoadLevel (5);//sin canaleta
		}
	}
}
public void juego2()//horizontal
{
	if (repeticiones != 0)
	{
		if (ej2_can == false)
		{   orientacion=0;
			Application.LoadLevel (4);//con canaleta
		}
		else
		{   orientacion=3;
			Application.LoadLevel (3);//sin canaleta
		}
```

```
		}
	}
	public void juego3()//combinado
	{
		if (repeticiones != 0)
		{
			if (ej3_can == false)
			{
				orientacion=2;
				Application.LoadLevel (6);//con canaleta
			}
			else
			{
				orientacion=4;
				Application.LoadLevel (7);//sin canaleta
			}
		}
	}
	public void check_1()
	{
		ej1_espejo = check1.isOn;
		if (ej1_espejo == false)
			ej1_.sprite = ej1m;
		else
			ej1_.sprite = ej1_esp;
	}
	public void check_2()
	{
		ej2_espejo = check2.isOn;
		if (ej2_espejo == false)
			ej2_.sprite = ej2m;
		else
			ej2_.sprite = ej2_esp;
	}
	public void check_3()
	{
		ej3_espejo = check3.isOn;
		if (ej3_espejo == false)
			ej3_.sprite = ej3m;
		else
			ej3_.sprite = ej3_esp;
	}
	///con o sin canaleta
	public void cscan_1()
	{
		ej1_can = cs1.isOn;
		if (ej1_can == false)
			ej2_.sprite = ej2m;
		else
			ej2_.sprite = ej2_esp;
	}
	public void cscan_2()
	{
		ej2_can = cs2.isOn;
		if (ej2_can == false)
			ej1_.sprite = ej1m;
		else
			ej1_.sprite = ej1_esp;
	}
	public void cscan_3()
	{
		ej3_can = cs3.isOn;
		if (ej3_can == false)
			ej3_.sprite = ej3m;
		else
			ej3_.sprite = ej3_esp;
	}
}
```

- pelota_beh.cs: Este *script* contiene los métodos necesarios para el funcionamiento del programa en cualquiera de los modos posibles, ya sea en la dificultad con o sin canaleta. También contiene métodos para salir del programa, reiniciar la partida actual o bien regresar al menú de configuración.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System.Collections.Generic;
using UnityStandardAssets.CrossPlatformInput;
public class pelota_beh : MonoBehaviour
{
	Rigidbody pelota_rig;
	Vector3 mouse_mov,pelota_pos,pos_ini;
	Vector3[] pos_cam=new Vector3[2]{new Vector3(0.0f,11.76f,-3.5f),new Vector3(0.0f,11.76f,3.5f)};
	Vector3[] pos_camh=new Vector3[2]{new Vector3(0.0f,14.3f,-4.54f),new Vector3(0.0f,14.3f,6.77f)};
	public GameObject pelotaa;
	public Camera cam;
	Ray rayo;
	int choques,choques_aux,orien;
	int repe,can=0,primeravez=0;
	int repepp=0;
	RaycastHit lugar;
	public Text mensajes,Debugeer;
	SphereCollider esfer;
	GameObject vacio_;
	public GameObject menub,reinib,salirb,esferaprueba;
	MeshRenderer pel;
	int[] errores;
	Vector3[] pos_pelota=new Vector3[5]{new Vector3(-4.16f,1.467f,-7.689f),new Vector3(-4.16f,1.467f,-3.9f),new Vector3(-4.16f,1.467f,-0.04f),new Vector3(-4.16f,1.467f,3.9f),
		new Vector3(-4.16f,1.467f,7.67f)};
	Vector3[] pos_pelotah=new Vector3[5]{new Vector3(14.647f,3.683f,-7.37f),new Vector3(14.647f,3.683f,-3.61f),new Vector3(14.647f,3.683f,0.24f),new Vector3(14.647f,3.683f,4.14f),
		new Vector3(14.647f,3.683f,7.71f)};
	Vector3[] pos_pelotah_sc=new Vector3[5]{new Vector3(-3.27f,1.766f,-6.23f),new Vector3(-1.559f,1.766f,-6.23f),new Vector3(0.12f,1.766f,-6.23f),new Vector3(1.826f,1.766f,-6.23f),
		new Vector3(3.423f,1.766f,-6.23f)};
	Vector3[] pos_pelotav_sc=new Vector3[5]{new Vector3(6.15f,1.2f,-3.57f),new Vector3(3.06f,1.2f,-3.57f),new Vector3(-0.02f,1.2f,-3.57f),new Vector3(-3.06f,1.2f,-3.57f),
		new Vector3(-6.05f,1.2f,-3.57f)};
	Vector3 pos_pelotacom = new Vector3 (4.897f, 0.405f, -2.38f);
	Vector3 pos_pelotacom_sc = new Vector3 (3.313f, 0.602f, 6.751f);
	bool acabo=false,menuon=false,cuerpo_p=false;
	// Use this for initialization
	void Start ()
	{
		pel = pelotaa.GetComponent<MeshRenderer> ();
		vacio_ = GameObject.Find ("vacio");
		pos_ini=pelotaa.transform.position;
		pelota_rig = GetComponent<Rigidbody> ();
		esfer = GetComponent<SphereCollider> ();
		repe = vacio_.GetComponent<menu> ().repeticiones;
		orien = vacio_.GetComponent<menu> ().orientacion;Debug.Log (orien);
		errores=new int[repe];
		limpia_e ();
		menub.SetActive (false);
		reinib.SetActive (false);
		salirb.SetActive (false);
		//if(orien==1)
		//	cam.transform.position = pos_cam [0];
		//if(orien==0)
		//	cam.transform.position = pos_camh [0];
		repetir();
```

```
	}
	// Update is called once per frame
	void Update ()
	{
		if (transform.position.y < -3.0f)
		{
			if(repepp<repe)
			  repetir();
			else
			{
				mensajes.text="Ejercicio terminado\nTuviste: "+choques+" errores totales\n";
				for(int j=0;j<repe;j++)
				{
					mensajes.text+="Repeticion "+(j+1)+": "+errores[j]+" errores\n";
				}
				pelotaa.SetActive(false);
				can=0;
			}
		}
		On ();
		//pelota_rig.AddForce (new Vector3 (0.0f, 1000.0f, 0.0f));

	}
	void repetir()
	{
		pel.enabled = false;
		//mensajes.text = "Repeticion " + (repepp + 1);
		//StartCoroutine ("posiciona_cam");
		pel.enabled = true;
		reinicio ();
		mensajes.text = "";
		//if(acabo)
		//	StopCoroutine ("repetir");

	}
	IEnumerator posiciona_cam()
	{
		if(primeravez!=0)
		{
			if (can == 3)
			{
				acabo=true;
			}
			if (can == 0)
			{
				acabo=true;
			}
			yield return new WaitForSeconds (0.5f);
		}
	}
	 void On()
	{
		//////////////////////
		cuerpo_p = BodySourceView.cuerpo_presente;
		if (cuerpo_p)
		{//Debug.Log (BodySourceView.posmanoder);
			Vector3 pos_limpiador;Debugeer.text="Posmder="+BodySourceView.posmanoder;
			pos_limpiador=asignaPos(BodySourceView.posmanoder);
			pos_limpiador=new  Vector3(Mathf.Clamp  (pos_limpiador.x,-4.602f,4.28f),pos_limpiador.y,Mathf.Clamp
(pos_limpiador.z,-8.2f,8.53f));//restringir la posicion de la esponja a la ventana
			if(BodySourceView.posmanoder.x<0.0f)
				pos_limpiador.z+=1.1f;
			if(BodySourceView.posmanoder.x>0.0f)
				pos_limpiador.z-=1.3f;
			if(BodySourceView.posmanoder.x==0.0f)
				pos_limpiador.z+=1.1f;
			rayo.origin=new
Vector3(pos_limpiador.x,pos_ini.y+1.0f,pos_limpiador.z);esferaprueba.transform.position=rayo.origin;
```

```
                        rayo.direction=new Vector3 (0.0f, -1.0f, 0.0f);
                        Physics.Raycast (rayo, out lugar);
                        if(lugar.collider)
                        {
                        //Debugeer.text=lugar.collider.name;//Raycast para detectar si el trapo esta debajo
                        Debug.Log (lugar.collider.name);
                        if(lugar.collider.name=="pelota")//verificar que el trapo esta debajo de la mano derecha
                        {
                                pelota_pos=pos_limpiador;
                                pelota_pos.y = pos_ini.y;//0.4599102f;
                                pelotaa.transform.position = pelota_pos;
                        }
                        }
                }
}
public void salir()
{
        Application.Quit ();
}
public void reinicio()
{
        acabo = false;
        pelotaa.SetActive (false);
        esfer.enabled=true;
        mensajes.text=" ";
        //choques = 0;
        pelota_rig.isKinematic = true;
        //pelota_rig.isKinematic = false;
        pelotaa.gameObject.SetActive (true);
        if(orien==1)
                pelotaa.transform.position = pos_pelota[can];//vertical
        if(orien==0)
                pelotaa.transform.position = pos_pelotah[can];//horizontal
        if(orien==2)
                pelotaa.transform.position = pos_pelotacom;//combinado
        if(orien==3)
                pelotaa.transform.position = pos_pelotah_sc[can];//horizontal sin canaleta
        if(orien==5)
                pelotaa.transform.position = pos_pelotav_sc[can];//vertical sin canaleta
        if(orien==4)
                pelotaa.transform.position = pos_pelotacom_sc;//combinado sin canaleta
        Renderer aux;
        aux=GetComponent<Renderer>();
        aux.material.color=new Color(1.0f,1.0f,1.0f);
}
public void reini_manual()
{
        acabo = false;
        limpia_e ();
        repepp = 0;
        choques = 0;
        can = 0;
        primeravez = 0;
        /*if(orien==1)
                cam.transform.position = pos_cam [0];
        if(orien==0)
                cam.transform.position = pos_camh [0];*/
        reinicio ();
        repetir();
        despliega ();
}
public void menu()
{
        Destroy (vacio_);
        Application.LoadLevel (1);
}
public void despliega()//muestra los botones de reinicio salir y menu
{
```

```
		menuon = !menuon;
		menub.SetActive (menuon);
		salirb.SetActive (menuon);
		reinib.SetActive (menuon);
	}
	void OnTriggerEnter(Collider conque)
	{
		if (conque.name == "can1" || conque.name == "can2"||conque.name == "can3"||conque.name == "can4"||conque.name == "can5"||conque.name=="can6"||

conque.name=="can7"||conque.name=="can8"||conque.name=="can9"||conque.name=="can10"||conque.name=="can11"||conque.name=="can12"||conque.name=="can13"||conque.name=="can14"
		    ||conque.name=="can15"||conque.name=="can0")
		{
			choques++;
			errores[repepp]+=1;
			Renderer aux;
			aux=GetComponent<Renderer>();
			aux.material.color=new Color(1.0f,0.0f,0.0f);
			pelota_rig.isKinematic =false;
		}
		if (conque.name == "Cube"||conque.name == "Cube2"||conque.name == "Cube3"||conque.name == "Cube4"||conque.name == "Cube5")
		{
			repepp+=1;
			Debug.Log ("caer");
			pelota_rig.isKinematic =false;
			pelota_rig.AddForce(new Vector3(0.5f,0.0f,0.5f));
			esfer.enabled=false;
			if(can<4)
				can++;
			else
				can=0;
			primeravez=1;
		}
	}
	void OnTriggerStay(Collider conque)
	{
		pelota_rig.AddForce(new Vector3(0.5f,0.0f,0.5f));
	}
	void OnTriggerExit(Collider conque)
	{
		Renderer aux;
		aux=GetComponent<Renderer>();
		aux.material.color=new Color(1.0f,1.0f,1.0f);
		pelota_rig.isKinematic =true;
	}
	void limpia_e()//pone en ceros el arreglo de errores
	{
		for (int i=0; i<repe; i++)
		{
			errores[i]=0;
		}
	}
	Vector3 asignaPos(Vector3 posmanoder)//asigna la posicion correcta al borrador en la ventana segun el valor del kinect para mano derecha
	{
		Vector3 posfinal = Vector3.zero;
		float x1=0.0f, y1=0.0f, x2=0.0f, y2=0.0f;
		//posfinal.y = 0.28f;
		////X de kinect se asigna a Z del borrador
		if (posmanoder.x < -0.1f)
		{
			y1=-8.2f;x1=-0.4f;
			y2=0.0f;x2=-0.1f;
			posfinal.z=27.333f*posmanoder.x+2.733f;
			//posfinal.z=((posmanoder.x-x1)/(x2-x1))*(y2-y1)+y1;
		}
```

```
                else
                {
                        if(posmanoder.x>-0.1f)
                        {
                                y1=0.0f;x1=-0.1f;
                                y2=8.53f;x2=0.3f;
                                posfinal.z=21.325f*posmanoder.x+2.133f;
                                //posfinal.z=((posmanoder.x-x1)/(x2-x1))*(y2-y1)+y1;
                        }
                        else
                        {
                                posfinal.z=0.0f;
                        }
                }
                ////Z de kinect se asigna a X del borrador
                if (posmanoder.z < 1.1f)
                {
                        y1=-4.602f;x1=0.9f;
                        y2=0.0f;x2=1.1f;
                        posfinal.x=23.01f*posmanoder.z-25.311f;
                        //posfinal.x=((posmanoder.z-x1)/(x2-x1))*(y2-y1)+y1;
                }
                else
                {
                        if(posmanoder.z>1.1f)
                        {
                                y1=4.28f;x1=1.3f;
                                y2=0.0f;x2=1.1f;
                                posfinal.x=21.4f*posmanoder.z-23.54f;
                                //posfinal.x=((posmanoder.z-x1)/(x2-x1))*(y2-y1)+y1;
                        }
                        else
                        {
                                posfinal.x=0.0f;
                        }
                }
                return posfinal;
        }
}
```

- BodySourceView.cs: Este *script* forma parte de los paquetes que provee Microsoft para la integración del *Kinect* en Unity3D, mencionado en el apartado de “Proceso de desarrollo de los programas”. Se encarga de manejar las posiciones de cada parte de los usuarios detectados por el *Kinect*. Fue modificado para que se comunicara con el *script* pelota_beh.cs, para enviarle la posición actual de la mano derecha del usuario detectado por el *Kinect*.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using Kinect = Windows.Kinect;

public class BodySourceView : MonoBehaviour
{
	public Material BoneMaterial;
	public GameObject BodySourceManager;
	/// <summary>
	//public GameObject cubo;
	public static Vector3 posmanoder;
	public static bool cuerpo_presente=false;
	///
	private Dictionary<ulong, GameObject> _Bodies = new Dictionary<ulong, GameObject>();
	private BodySourceManager _BodyManager;

	private Dictionary<Kinect.JointType, Kinect.JointType> _BoneMap = new Dictionary<Kinect.JointType, Kinect.JointType>()
	{
		{ Kinect.JointType.FootLeft, Kinect.JointType.AnkleLeft },
		{ Kinect.JointType.AnkleLeft, Kinect.JointType.KneeLeft },
		{ Kinect.JointType.KneeLeft, Kinect.JointType.HipLeft },
		{ Kinect.JointType.HipLeft, Kinect.JointType.SpineBase },

		{ Kinect.JointType.FootRight, Kinect.JointType.AnkleRight },
		{ Kinect.JointType.AnkleRight, Kinect.JointType.KneeRight },
		{ Kinect.JointType.KneeRight, Kinect.JointType.HipRight },
		{ Kinect.JointType.HipRight, Kinect.JointType.SpineBase },

		{ Kinect.JointType.HandTipLeft, Kinect.JointType.HandLeft },
		{ Kinect.JointType.ThumbLeft, Kinect.JointType.HandLeft },
		{ Kinect.JointType.HandLeft, Kinect.JointType.WristLeft },
		{ Kinect.JointType.WristLeft, Kinect.JointType.ElbowLeft },
		{ Kinect.JointType.ElbowLeft, Kinect.JointType.ShoulderLeft },
		{ Kinect.JointType.ShoulderLeft, Kinect.JointType.SpineShoulder },

		{ Kinect.JointType.HandTipRight, Kinect.JointType.HandRight },
		{ Kinect.JointType.ThumbRight, Kinect.JointType.HandRight },
		{ Kinect.JointType.HandRight, Kinect.JointType.WristRight },
		{ Kinect.JointType.WristRight, Kinect.JointType.ElbowRight },
		{ Kinect.JointType.ElbowRight, Kinect.JointType.ShoulderRight },
		{ Kinect.JointType.ShoulderRight, Kinect.JointType.SpineShoulder },

		{ Kinect.JointType.SpineBase, Kinect.JointType.SpineMid },
		{ Kinect.JointType.SpineMid, Kinect.JointType.SpineShoulder },
		{ Kinect.JointType.SpineShoulder, Kinect.JointType.Neck },
		{ Kinect.JointType.Neck, Kinect.JointType.Head },
	};

	void Update ()
	{
		if (BodySourceManager == null)
		{
			return;
		}

		_BodyManager = BodySourceManager.GetComponent<BodySourceManager>();
		if (_BodyManager == null)
		{
			return;
		}

		Kinect.Body[] data = _BodyManager.GetData();
		if (data == null)
		{
			return;
		}
		//Debug.Log ("cuerpos presentes="+data.Length);
```

```
		List<ulong> trackedIds = new List<ulong>();
		foreach(var body in data)
		{
			if (body == null)
			{cuerpo_presente=false;
				continue;
			}

			if(body.IsTracked)
			{cuerpo_presente=true;
				trackedIds.Add (body.TrackingId);
			}
		}

		List<ulong> knownIds = new List<ulong>(_Bodies.Keys);

		// First delete untracked bodies
		foreach(ulong trackingId in knownIds)
		{
			if(!trackedIds.Contains(trackingId))
			{
				Destroy(_Bodies[trackingId]);
				_Bodies.Remove(trackingId);
			}
		}

		foreach(var body in data)
		{
			if (body == null)
			{
				continue;
			}

			if(body.IsTracked)
			{
				if(!_Bodies.ContainsKey(body.TrackingId))
				{
					_Bodies[body.TrackingId] = CreateBodyObject(body.TrackingId);
				}

				RefreshBodyObject(body, _Bodies[body.TrackingId]);
			}
		}
}

private GameObject CreateBodyObject(ulong id)
{
		GameObject body = new GameObject("Body:" + id);

		for (Kinect.JointType jt = Kinect.JointType.SpineBase; jt <= Kinect.JointType.ThumbRight; jt++)
		{
			GameObject jointObj = GameObject.CreatePrimitive(PrimitiveType.Cube);

			LineRenderer lr = jointObj.AddComponent<LineRenderer>();
			lr.SetVertexCount(2);
			lr.material = BoneMaterial;
			lr.SetWidth(0.05f, 0.05f);

			jointObj.transform.localScale = new Vector3(0.3f, 0.3f, 0.3f);
			jointObj.name = jt.ToString();
			jointObj.transform.parent = body.transform;
			jointObj.GetComponent<Renderer>().enabled=false;
		}

		return body;
}

private void RefreshBodyObject(Kinect.Body body, GameObject bodyObject)
```

```
	{
		for (Kinect.JointType jt = Kinect.JointType.SpineBase; jt <= Kinect.JointType.ThumbRight; jt++)
		{
			Kinect.Joint sourceJoint = body.Joints[jt];
			Kinect.Joint? targetJoint = null;

			if(_BoneMap.ContainsKey(jt))
			{
				targetJoint = body.Joints[_BoneMap[jt]];
			}

			Transform jointObj = bodyObject.transform.FindChild(jt.ToString());
			jointObj.localPosition = GetVector3FromJoint(sourceJoint);

			LineRenderer lr = jointObj.GetComponent<LineRenderer>();
			if(targetJoint.HasValue)
			{

	if(jt==Kinect.JointType.HandRight&&jointObj.localPosition.z>0.7f&&jointObj.localPosition.z<1.7f)
				{
					//cubo.transform.localPosition=jointObj.localPosition;
					posmanoder=jointObj.localPosition;
				}
				//lr.SetPosition(0, jointObj.localPosition);
				//lr.SetPosition(1, GetVector3FromJoint(targetJoint.Value));
				//lr.SetColors(GetColorForState                    (sourceJoint.TrackingState),
GetColorForState(targetJoint.Value.TrackingState));
			}
			else
			{
				lr.enabled = false;
			}
		}
	}

	private static Color GetColorForState(Kinect.TrackingState state)
	{
		switch (state)
		{
		case Kinect.TrackingState.Tracked:
			return Color.green;

		case Kinect.TrackingState.Inferred:
			return Color.red;

		default:
			return Color.black;
		}
	}

	private static Vector3 GetVector3FromJoint(Kinect.Joint joint)
	{
		return new Vector3(joint.Position.X , joint.Position.Y, joint.Position.Z);
		//return new Vector3(joint.Position.X * 4, joint.Position.Y * 10, joint.Position.Z * 4);
	}
}
```

Apéndice 4

A continuación se muestran los *scripts* correspondientes al programa de limpiar la ventana para sensor *Kinect*.

- log.cs: Este *script* contiene los métodos necesarios para ingresar al programa o bien salir del mismo.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class log : MonoBehaviour
{
	public InputField nombre_v;
	string nombre_usuario;
	// Use this for initialization
	void Start ()
	{

	}

	// Update is called once per frame
	void Update ()
	{
		if (Input.GetKeyDown (KeyCode.KeypadEnter) || Input.GetAxis("Submit")>0.0f)
		{Debug.Log ("enter");
			log_in();
		}
	}
	public void log_in()//sirve para ingresar al juego
	{
		if (nombre_v.text != "")//si no se ingresa nada no se puede acceder
		{

			Application.LoadLevel(1);
		}
		else
		{
			Debug.Log ("No se ingreso nada");
		}
	}
	public void salir()//para salir del programa con boton salir
	{
		Application.Quit ();
	}
}
```

- menu_beh_.cs: este *script* contiene los métodos que permiten ingresar a cualquiera de los tres modos del programa e indicar el número de repeticiones deseada (el número de manchas de suciedad que habrá en la ventana para limpiarlas). Adicionalmente contiene métodos para salir del programa o bien regresar a la pantalla de *log in*.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
public class menu_beh_ : MonoBehaviour
{
	public Toggle check;
	public InputField repeticions_;
	//bool espejo;
	public int repe,modo;//los modos son: 0-horizontal 1-vertical 2-libre
	//public Vector3[] ptos_kinect=new Vector3[5]{Vector3.zero,Vector3.zero,Vector3.zero,Vector3.zero,Vector3.zero};
	// Use this for initialization
	void Start ()
	{
		//espejo = false;

	}
	void Awake()//indicamos que no se destruya vacio, el cual contiene un script con la variable del
	{//numero de repeticiones ingresadas
		DontDestroyOnLoad (this);
	}
	// Update is called once per frame
	void Update ()
	{
		if (repeticions_.text != "")//solo se intenta pasar a enter si hay algo escrito
		{
			System.Int32.TryParse (repeticions_.text, out repe);
			if(repe==0)//si se ingresa algo que no sea entero, se limpia el panel y no se puede ingresar
			{
				repeticions_.text="";
			}
			if(repe>25)
			{
				repe=25;
			}
		}
	}
	public void regresa()
	{
		Destroy (this);
		Application.LoadLevel (0);
	}
	public void sal()
	{
		Application.Quit ();
	}
    public void horiz()//modo horizontal
	{
		if (repe > 0&&repe<=25)
		{
			modo = 0;
			Application.LoadLevel (2);
		}
	}
	public void verti()//modo vertical
	{
		if (repe > 0&&repe<=25)
		{
			modo = 1;
			Application.LoadLevel (2);
		}
	}
	public void libre()//modo en circulos
	{
		if (repe > 0&&repe<=25)
		{
				modo=2;
				Application.LoadLevel (2);
		}
```

```
	}
}
```

- mov_limpiador.cs: Este *script* contiene los métodos necesarios para el movimiento de la esponja y todas las comprobaciones necesarias para el funcionamiento del programa. También contiene un método para reiniciar la partida.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

//////////////////////
////
///
///

public class mov_limpiador : MonoBehaviour
{
	TuioInput touch_man;
	Ray rayo,rayoaux;
	public Camera cam;
	RaycastHit lugar,lugaraux;
	Vector3 lim_pos;
	Vector3[] cache=new Vector3[2];
//	Vector3 resul;
	float y_ini;
	LineRenderer rastro;
	int semaforo=0,indice=0,contador=0;
	GameObject menu;
	public Text direccion,Debugeer;
//	Object aux_o1,aux_o2;
	int repeticiones,posicion;
	string[] dir=new string[2];
	bool[] esp_t=new bool[3];//indicadores de acabar de limpiar la barra de espuma
	bool[] det=new bool[25];
	bool corretiempo,terminado=false,flag=false,cuerpo_p;
	public GameObject[] manchas=new GameObject[15];
	float tiempo=0.0f;
	public Text cronometro;
	public GameObject reini_b,salir_b,regres_b;
	Renderer[] espuma_rend=new Renderer[25]{null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null};
	Vector3[] posmugre_hor=new Vector3[25]{new Vector3(0.453f,0.25f,0.696f),new Vector3(0.451f,0.25f,0.517f),new Vector3(0.449f,0.25f,0.279f),new Vector3(0.44f,0.25f,0.0f),new Vector3(0.443f,0.25f,-0.256f),
		new Vector3(0.431f,0.25f,-0.559f),new Vector3(0.416f,0.25f,-0.83f),new Vector3(0.395f,0.25f,-1.032f),new Vector3(0.061f,0.25f,0.948f),new Vector3(0.013f,0.25f,0.65f),
		new Vector3(0.012f,0.25f,0.38f),new Vector3(0.01f,0.25f,0.098f),new Vector3(0.019f,0.25f,-0.187f),new Vector3(0.025f,0.25f,-0.473f),new Vector3(0.015f,0.25f,-0.766f),
		new Vector3(0.003f,0.25f,-1.046f),new Vector3(-0.38f,0.25f,0.966f),new Vector3(-0.392f,0.25f,0.685f),new Vector3(-0.384f,0.25f,0.397f),new Vector3(-0.396f,0.25f,0.108f),
		new Vector3(-0.395f,0.25f,-0.177f),new Vector3(-0.396f,0.25f,-0.464f),new Vector3(-0.397f,0.25f,-0.75f),new Vector3(-0.4f,0.25f,-0.916f),new Vector3(-0.405f,0.25f,-1.0f)};

	Vector3[] posmugre_cir=new Vector3[25]{new Vector3(0.486f,0.25f,0.57f),new Vector3(0.205f,0.25f,0.696f),new Vector3(-0.1f,0.25f,0.914f),new Vector3(-0.402f,0.25f,0.8f),new Vector3(-0.444f,0.25f,0.407f),
		new Vector3(-0.475f,0.25f,0.015f),new Vector3(-0.487f,0.25f,-0.376f),new Vector3(-0.326f,0.25f,-0.775f),new Vector3(-0.037f,0.25f,-0.908f),new Vector3(0.275f,0.25f,-0.905f),
		new Vector3(0.47f,0.25f,-0.548f),new Vector3(0.47f,0.25f,-0.15f),new Vector3(0.243f,0.25f,0.088f),new Vector3(-0.027f,0.25f,0.032f),new Vector3(0.06f,0.25f,-0.365f),
		new Vector3(0.488f,0.25f,-0.321f),new Vector3(0.212f,0.25f,-0.203f),new Vector3(-0.055f,0.25f,-0.324f),new Vector3(-0.307f,0.25f,-0.316f),new Vector3(-0.489f,0.25f,-0.314f),
```

```
		new Vector3(0.459f,0.25f,-0.915f),new Vector3(0.211f,0.25f,-0.608f),new Vector3(-0.046f,0.25f,-0.852f),new
Vector3(-0.287f,0.25f,-0.884f),new Vector3(-0.451f,0.25f,-0.625f)};
	float till=0.0f;

	float[] bes=new float[4]{0.0f,0.0f,0.0f,0.0f};
	float[] Mes=new float[4]{0.0f,0.0f,0.0f,0.0f};
	// Use this for initialization
	void Start ()
	{

		multitoque_init ();//inicializar funciones multitoque

		direccion.text = "";
		dir [0] = "";
		dir [1] = "";
		for (int i=0; i<25; i++)
		{
			det[i]=false;
		}
		for (int i=0; i<3; i++)
		{
			esp_t[i]=false;
		}
		menu = GameObject.Find ("menu_beh");
		rastro = GetComponent<LineRenderer> ();
		y_ini = 0.28f;
		repeticiones = menu.GetComponent<menu_beh_> ().repe;
		posicion = menu.GetComponent<menu_beh_> ().modo;
//		ESP_KINECT = menu.GetComponent<menu_beh_> ().ptos_kinect;//obtener puntos de calibracion de kinect
		calcular_regresiones ();//calcular ecuaciones lineales para pasar del espacio del kinect al espacio de la ventana
		//Debug.Log (repeticiones);
		//Debug.Log (posicion);
		for (int i=0; i<25; i++)
		{
			manchas[i].SetActive(false);
		}
		creaespuma ();
		for (int i=0; i<25; i++)
		{
			espuma_rend[i]=manchas[i].GetComponent<Renderer>();
		}
	}

	// Update is called once per frame
	void Update ()
	{//Debug.Log (Input.mousePosition);
		//bool cache = false;
		if (corretiempo)
			tiempo += Time.deltaTime;
		//Debug.Log (tiempo);
		//Input.multiTouchEnabled = true;

		for (int i=0; i<repeticiones; i++)//si ya se quito toda la mugre se termina el juego
		{
			if(i==0)
				terminado=det[0];
			else
				terminado=terminado&&det[i];
		}
		if (terminado)//comprobacion de terminacion
		{
			corretiempo=false;
			Debug.Log ("termino");
			float tiempo_truncado=Mathf.Round(tiempo*100.0f)/100.0f;
			cronometro.text="Tiempo total : "+tiempo_truncado+" segundos ";
			gameObject.SetActive(false);
		}
		////movimiento de la textura de la mugre
```

```
		if (till < 0.05f)
			till += 0.0001f * Time.deltaTime;
		if (posicion == 0)//modo horizontal
		{
			for (int i=0; i<25; i++)
			{
				espuma_rend [i].material.mainTextureOffset = new Vector2 (0.0f, till);
			}
		}
		if (posicion == 1)//modo vertical
		{
			for (int i=0; i<25; i++)
			{
				espuma_rend [i].material.mainTextureOffset = new Vector2 (till, 0.0f);
			}
		}
		On ();
	}
	void On()//originalmente onmouseover
	{
		cuerpo_p = BodySourceView.cuerpo_presente;
		if (cuerpo_p)
		{Debug.Log (BodySourceView.posmanoder);
			Vector3 pos_limpiador;Debugeer.text="Posmder="+BodySourceView.posmanoder;
			pos_limpiador=asignaPos(BodySourceView.posmanoder);
			pos_limpiador=new Vector3(Mathf.Clamp (pos_limpiador.x,-0.503f,0.503f),pos_limpiador.y,Mathf.Clamp
(pos_limpiador.z,-1.035f,1.035f));//restringir la posicion de la esponja a la ventana
			///
			rayo.origin=new Vector3(pos_limpiador.x,pos_limpiador.y+0.01f,pos_limpiador.z);
			rayo.direction=new Vector3 (0.0f, -1.0f, 0.0f);
			Physics.Raycast (rayo, out lugar);//Debugeer.text=lugar.collider.name;//Raycast para detectar si el trapo
esta debajo
			if(lugar.collider.name=="limpiador")//verificar que el trapo esta debajo de la mano derecha
			{///
				transform.position=pos_limpiador;
				rayoaux.origin = transform.position;
				rayoaux.direction = new Vector3 (0.0f, -1.0f, 0.0f);
				Physics.Raycast (rayoaux, out lugaraux);//Raycast para detectar si hay manchas de suciedad
debajo del trapo
				if (lugaraux.collider != null && lugaraux.collider.name != "ventana" && lugaraux.collider.name
!= "limpiador")
				{
					corretiempo = true;
					if(contador<repeticiones)
					{
						int indice=contador+1;
						string	nombre="espuma"+indice;//direccion.text=nombre;//Debug.Log
(nombre);
						if (lugaraux.collider.name == nombre)
						{
							manchas [contador].SetActive (false);
							det [contador] = true;
							contador++;
						}
					}
				}
			}
		}
		/*
		float distancia_t=0.0f;
		int toques = gameObject.GetComponent<TuioInput> ().num_toques;
		Vector2[] toques_pos = new Vector2[2];

		if (toques == 1)
		{//Debug.Log ("un toque");
			toques_pos[0]=gameObject.GetComponent<TuioInput> ().toque12[0];
			rayo = cam.ScreenPointToRay (new Vector3(toques_pos[0].x,toques_pos[0].y,0.0f));
		}
```

```
		if (toques == 2)
		{
			Vector2 promedio;
			toques_pos[0]=gameObject.GetComponent<TuioInput> ().toque12[0];
			toques_pos[1]=gameObject.GetComponent<TuioInput> ().toque12[1];
			distancia_t=Vector2.Distance(toques_pos[0],toques_pos[1]);
			float	d_toque1_pelota=Vector3.Distance(transform.position,Camera.main.ScreenToWorldPoint(new
Vector3(toques_pos[0].x,toques_pos[0].y,1.038f)));
			float	d_toque2_pelota=Vector3.Distance(transform.position,Camera.main.ScreenToWorldPoint(new
Vector3(toques_pos[1].x,toques_pos[1].y,1.038f)));
			//direccion.text="toque 1 a:"+d_toque1_pelota+"de distancia";
			//direccion.text+="toque 2 a:"+d_toque2_pelota+"de distancia";
			if(distancia_t<120.0f)
			{
			promedio=(toques_pos[0]+toques_pos[1])/2;
			rayo = cam.ScreenPointToRay (new Vector3(promedio.x,promedio.y,0.0f));
			}
			else
			{
				if(d_toque1_pelota<d_toque2_pelota)
					rayo		=		cam.ScreenPointToRay		(new
Vector3(toques_pos[0].x,toques_pos[0].y,0.0f));
				if(d_toque2_pelota<d_toque1_pelota)
					rayo		=		cam.ScreenPointToRay		(new
Vector3(toques_pos[1].x,toques_pos[1].y,0.0f));

			}
		}
		if (toques==1||(toques==2))
	{
			Physics.Raycast (rayo, out lugar);Debugeer.text=lugar.collider.name;//Raycast para definir posicion del
trapo
			if(lugar.collider.name=="limpiador")
		 {
			lim_pos = lugar.point;
			lim_pos.y = y_ini;
			transform.position = lim_pos;
			rayoaux.origin = transform.position;
			rayoaux.direction = new Vector3 (0.0f, -1.0f, 0.0f);
			Physics.Raycast (rayoaux, out lugaraux);//Raycast para detectar si hay manchas de suciedad debajo
del trapo
			//direccion.text=lugaraux.collider.name;//Debug.Log (lugaraux.collider.name);
			if (lugaraux.collider != null && lugaraux.collider.name != "ventana" && lugaraux.collider.name !=
"limpiador")
			{
				corretiempo = true;
				if(contador<repeticiones)
				{
					int indice=contador+1;
					string nombre="espuma"+indice;//direccion.text=nombre;//Debug.Log (nombre);
					if (lugaraux.collider.name == nombre)
					{
						manchas [contador].SetActive (false);
						det [contador] = true;
						contador++;
					}
				}
			}
		 }
		}*/
	}
	void creaespuma()
	{
		if (posicion == 0)
		{//se eligio modo horizontal
			Quaternion auxi_rot=Quaternion.identity;
			for(int i=0;i<repeticiones;i++)
			{
```

```
                                manchas[i].transform.position=posmugre_hor[i];
                                auxi_rot=manchas[i].transform.localRotation;
                                auxi_rot.eulerAngles=new Vector3(90.0f,270.0f,0.0f);
                                manchas[i].transform.localRotation=auxi_rot;
                        }
                }
                if (posicion == 1)//se eligio modo vertical
                {
                        //se deja tal cual estan las manchas
                }
                if (posicion == 2)
                {//se eligio modo en circulos
                        int aux_rep=0;
                        if(repeticiones>15)
                        {
                                aux_rep=15;
                                repeticiones=aux_rep;
                        }
                        else
                                aux_rep=repeticiones;

                        for(int i=0;i<aux_rep;i++)
                        {
                                manchas[i].transform.position=posmugre_cir[i];
                        }
                        for (int i=0; i<aux_rep; i++)
                        {
                                manchas [i].SetActive (true);
                        }
                }
                if (posicion == 0 || posicion == 1)
                {
                        for (int i=0; i<repeticiones; i++)
                        {
                                manchas [i].SetActive (true);
                        }
                }
}
public void reinicio()
{
        rastro.SetVertexCount(0);
        corretiempo = false;
        tiempo = 0.0f;
        cronometro.text = "";
        contador = 0;
        till = 0.0f;
        for (int i=0; i<25; i++)
        {
                det[i]=false;
        }
        for (int i=0; i<3; i++)
        {
                esp_t[i]=false;
        }
        transform.position = new Vector3 (0.549f, 0.28f, 1.038f);
        for (int i=0; i<25; i++)
        {
                manchas[i].SetActive(false);
        }
        creaespuma ();
        gameObject.SetActive(true);
        replie_desplie ();
}
void multitoque_init()
{
        Input.multiTouchEnabled = true;
}
public void replie_desplie()//repliega o despliega el menu
```

```
{
	flag = !flag;
	reini_b.SetActive (flag);
	salir_b.SetActive (flag);
	regres_b.SetActive (flag);
}
void calcular_regresiones()//regresiones lineales para pasar del espacio del kinect al de la ventana
{
	//calcular el valor de m de ecuacion 1

	//calcular el valor de b de ecuacion 1

	//calcular el valor de m de ecuacion 2

	//calcular el valor de b de ecuacion 2

	//calcular el valor de m de ecuacion 3

	//calcular el valor de b de ecuacion 3

	//calcular el valor de m de ecuacion 4

	//calcular el valor de b de ecuacion 4

}
Vector3 asignaPos(Vector3 posmanoder)//asigna la posicion correcta al borrador en la ventana segun el valor del kinect para mano derecha
{
	Vector3 posfinal = Vector3.zero;
	float x1=0.0f, y1=0.0f, x2=0.0f, y2=0.0f;
	posfinal.y = 0.28f;
	////X de kinect se asigna a Z del borrador
	if (posmanoder.x < -0.1f)
	{
		y1=1.035f;x1=-0.4f;
		y2=0.0f;x2=-0.1f;
		posfinal.z=-3.45f*posmanoder.x-0.345f;
		//posfinal.z=((posmanoder.x-x1)/(x2-x1))*(y2-y1)+y1;
	}
	else
	{
		if(posmanoder.x>-0.1f)
		{
			y1=0.0f;x1=-0.1f;
			y2=-1.035f;x2=0.3f;
			posfinal.z=-2.587f*posmanoder.x-0.259f;
			//posfinal.z=((posmanoder.x-x1)/(x2-x1))*(y2-y1)+y1;
		}
		else
		{
			posfinal.z=0.0f;
		}
	}
	////Z de kinect se asigna a X del borrador
	if (posmanoder.z < 1.1f)
	{
		y1=0.503f;x1=0.9f;
		y2=0.0f;x2=1.1f;
		posfinal.x=-2.515f*posmanoder.z+2.767f;
		//posfinal.x=((posmanoder.z-x1)/(x2-x1))*(y2-y1)+y1;
	}
	else
	{
		if(posmanoder.z>1.1f)
		{
			y1=-0.503f;x1=1.3f;
			y2=0.0f;x2=1.1f;
			posfinal.x=-2.515f*posmanoder.z+2.716f;
```

```
                                        //posfinal.x=((posmanoder.z-x1)/(x2-x1))*(y2-y1)+y1;
                                }
                                else
                                {
                                        posfinal.x=0.0f;
                                }
                        }
                        return posfinal;
                }
}
```

- BodySourceView.cs: Este *script* forma parte de los paquetes que provee Microsoft para la integración del *Kinect* en Unity3D, mencionado en el apartado de Proceso de desarrollo de los programas. Se encarga de manejar las posiciones de cada parte de los usuarios detectados por el *Kinect*. Fue modificado para que se comunicara con el *script* mov_limpiador.cs, para enviarle la posición actual de la mano derecha del usuario detectado por el *Kinect*.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using Kinect = Windows.Kinect;

public class BodySourceView : MonoBehaviour
{
        public Material BoneMaterial;
        public GameObject BodySourceManager;
        /// <summary>
        //public GameObject cubo;
        public static Vector3 posmanoder;
        public static bool cuerpo_presente=false;
        ///
        private Dictionary<ulong, GameObject> _Bodies = new Dictionary<ulong, GameObject>();
        private BodySourceManager _BodyManager;

        private Dictionary<Kinect.JointType, Kinect.JointType> _BoneMap = new Dictionary<Kinect.JointType, Kinect.JointType>()
        {
                { Kinect.JointType.FootLeft, Kinect.JointType.AnkleLeft },
                { Kinect.JointType.AnkleLeft, Kinect.JointType.KneeLeft },
                { Kinect.JointType.KneeLeft, Kinect.JointType.HipLeft },
                { Kinect.JointType.HipLeft, Kinect.JointType.SpineBase },

                { Kinect.JointType.FootRight, Kinect.JointType.AnkleRight },
                { Kinect.JointType.AnkleRight, Kinect.JointType.KneeRight },
                { Kinect.JointType.KneeRight, Kinect.JointType.HipRight },
                { Kinect.JointType.HipRight, Kinect.JointType.SpineBase },

                { Kinect.JointType.HandTipLeft, Kinect.JointType.HandLeft },
                { Kinect.JointType.ThumbLeft, Kinect.JointType.HandLeft },
                { Kinect.JointType.HandLeft, Kinect.JointType.WristLeft },
                { Kinect.JointType.WristLeft, Kinect.JointType.ElbowLeft },
                { Kinect.JointType.ElbowLeft, Kinect.JointType.ShoulderLeft },
                { Kinect.JointType.ShoulderLeft, Kinect.JointType.SpineShoulder },

                { Kinect.JointType.HandTipRight, Kinect.JointType.HandRight },
                { Kinect.JointType.ThumbRight, Kinect.JointType.HandRight },
                { Kinect.JointType.HandRight, Kinect.JointType.WristRight },
                { Kinect.JointType.WristRight, Kinect.JointType.ElbowRight },
```

```
        { Kinect.JointType.ElbowRight, Kinect.JointType.ShoulderRight },
        { Kinect.JointType.ShoulderRight, Kinect.JointType.SpineShoulder },

        { Kinect.JointType.SpineBase, Kinect.JointType.SpineMid },
        { Kinect.JointType.SpineMid, Kinect.JointType.SpineShoulder },
        { Kinect.JointType.SpineShoulder, Kinect.JointType.Neck },
        { Kinect.JointType.Neck, Kinect.JointType.Head },
};

void Update ()
{
        if (BodySourceManager == null)
        {
                return;
        }

        _BodyManager = BodySourceManager.GetComponent<BodySourceManager>();
        if (_BodyManager == null)
        {
                return;
        }

        Kinect.Body[] data = _BodyManager.GetData();
        if (data == null)
        {
                return;
        }
        //Debug.Log ("cuerpos presentes="+data.Length);
        List<ulong> trackedIds = new List<ulong>();
        foreach(var body in data)
        {
                if (body == null)
                {cuerpo_presente=false;
                        continue;
                }

                if(body.IsTracked)
                {cuerpo_presente=true;
                        trackedIds.Add (body.TrackingId);
                }
        }

        List<ulong> knownIds = new List<ulong>(_Bodies.Keys);

        // First delete untracked bodies
        foreach(ulong trackingId in knownIds)
        {
                if(!trackedIds.Contains(trackingId))
                {
                        Destroy(_Bodies[trackingId]);
                        _Bodies.Remove(trackingId);
                }
        }

        foreach(var body in data)
        {
                if (body == null)
                {
                        continue;
                }

                if(body.IsTracked)
                {
                        if(!_Bodies.ContainsKey(body.TrackingId))
                        {
                                _Bodies[body.TrackingId] = CreateBodyObject(body.TrackingId);
                        }
```

```
                                        RefreshBodyObject(body, _Bodies[body.TrackingId]);
                                }
                        }
                }

                private GameObject CreateBodyObject(ulong id)
                {
                        GameObject body = new GameObject("Body:" + id);

                        for (Kinect.JointType jt = Kinect.JointType.SpineBase; jt <= Kinect.JointType.ThumbRight; jt++)
                        {
                                GameObject jointObj = GameObject.CreatePrimitive(PrimitiveType.Cube);

                                LineRenderer lr = jointObj.AddComponent<LineRenderer>();
                                lr.SetVertexCount(2);
                                lr.material = BoneMaterial;
                                lr.SetWidth(0.05f, 0.05f);

                                jointObj.transform.localScale = new Vector3(0.3f, 0.3f, 0.3f);
                                jointObj.name = jt.ToString();
                                jointObj.transform.parent = body.transform;
                                jointObj.GetComponent<Renderer>().enabled=false;
                        }

                        return body;
                }

                private void RefreshBodyObject(Kinect.Body body, GameObject bodyObject)
                {
                        for (Kinect.JointType jt = Kinect.JointType.SpineBase; jt <= Kinect.JointType.ThumbRight; jt++)
                        {
                                Kinect.Joint sourceJoint = body.Joints[jt];
                                Kinect.Joint? targetJoint = null;

                                if(_BoneMap.ContainsKey(jt))
                                {
                                        targetJoint = body.Joints[_BoneMap[jt]];
                                }

                                Transform jointObj = bodyObject.transform.FindChild(jt.ToString());
                                jointObj.localPosition = GetVector3FromJoint(sourceJoint);

                                LineRenderer lr = jointObj.GetComponent<LineRenderer>();
                                if(targetJoint.HasValue)
                                {

                if(jt==Kinect.JointType.HandRight&&jointObj.localPosition.z>0.9f&&jointObj.localPosition.z<1.6f)
                                        {
                                                //cubo.transform.localPosition=jointObj.localPosition;
                                                posmanoder=jointObj.localPosition;
                                        }
                                        //lr.SetPosition(0, jointObj.localPosition);
                                        //lr.SetPosition(1, GetVector3FromJoint(targetJoint.Value));
                                        //lr.SetColors(GetColorForState                                   (sourceJoint.TrackingState),
GetColorForState(targetJoint.Value.TrackingState));
                                }
                                else
                                {
                                        lr.enabled = false;
                                }
                        }
                }

                private static Color GetColorForState(Kinect.TrackingState state)
                {
                        switch (state)
                        {
                        case Kinect.TrackingState.Tracked:
```

```
                return Color.green;

            case Kinect.TrackingState.Inferred:
                return Color.red;

            default:
                return Color.black;
            }
        }

        private static Vector3 GetVector3FromJoint(Kinect.Joint joint)
        {
            return new Vector3(joint.Position.X , joint.Position.Y, joint.Position.Z);
            //return new Vector3(joint.Position.X * 4, joint.Position.Y * 10, joint.Position.Z * 4);
        }
}
```

- regre_sal.cs: Este *script* contiene únicamente métodos para regresar al menú de configuración o bien salir del programa.

```
using UnityEngine;
using System.Collections;
public class regre_sal : MonoBehaviour
{
        GameObject menuu;
        void Start()
        {
            menuu = GameObject.Find ("menu_beh");
        }
        public void regresar()
        {
            Destroy (menuu);
            Application.LoadLevel (1);
        }
        public void salir()
        {
            Application.Quit ();
        }
}
```

Glosario

-Aceleradora gráfica: una aceleradora gráfica es un dispositivo diseñado para procesar principalmente todas las operaciones del *pipeline* gráfico que nos permiten desplegar gráficos en una computadora. Aunque originalmente su fin era solo el de procesar operaciones del *pipeline* gráfico, en la actualidad también se utilizan para realizar operaciones complejas de punto flotante enfocadas a cálculos matemáticos, y científicos, así como programas diseñados para procesarse en paralelo (CUDA y Open CL). Adicionalmente, prácticamente cualquier aceleradora gráfica moderna sea integrada o dedicada, incluye decodificación y/o codificación de video vía *hardware* de uno o más estándares como H.264, H.265, MPEG-2, etc. Lo cual permite que la CPU se limite a solicitar a la aceleradora la decodificación del video pero sin ocupar recursos para procesar dicho archivo de video relegándolo completamente al *hardware* de la aceleradora gráfica. **[23]**

De manera análoga a la arquitectura Von Newman, cuentan con una GPU o unidad de procesamiento gráfico, una memoria de trabajo principal (A veces llamada VRAM), un grupo de buses que comunican el sistema, un medio de comunicación con la tarjeta madre (principalmente con el estándar PCI Express) además de un convertidor analógico digital. Una particularidad de una aceleradora gráfica es que en realidad la GPU, cuenta con un número determinado de unidades de ejecución trabajando simultáneamente y procesando en paralelo, llamadas *CUDA cores* por NVIDIA o *Streaming processors* por AMD. Todas las unidades tienen comunicación con la memoria de trabajo y se pueden comunicar entre sí. Van desde unas pocas (2 a 10) en dispositivos móviles Android y iOS, hasta más de 3000 en el caso de aceleradoras gráficas de gama muy alta. En las tarjetas modernas, cualquier unidad de ejecución puede encargarse de cualquier tipo de operación del *pipeline* gráfico que se le solicite, por lo que siempre estarán trabajando en paralelo pero no necesariamente ejecutando la misma tarea. Es por esto que en condiciones normales, entre mayor sea el número de unidades de ejecución y mayor sea la frecuencia de reloj a la que trabajen, mayor será el poder de procesamiento gráfico. Lo cual se traduce simple y sencillamente en un *framerate* mayor para cualquier programa gráfico que haga uso de la aceleradora. O bien en la capacidad de la tarjeta para utilizar técnicas más complejas o demandantes sobre los gráficos que se está *renderizando*.

-API Gráfica: API viene de las siglas en inglés de *Application Programming Interface*, es decir, interfaz de programación de aplicaciones. Se trata de un conjunto de subrutinas, protocolos, funciones, métodos y herramientas para construir aplicaciones y *software*. **[3]**

La idea esencial de una API es que esta funciona como una interfaz entre operaciones de medio o bajo nivel y uno o varios lenguajes de programación, generalmente de alto

nivel. En el caso particular de las API gráficas, estos funcionan como una interfaz entre el *hardware* de la computadora que permite el despliegue de gráficos y un lenguaje de programación de alto nivel, generalmente C++, C#, Python, Javascript, entre otros. De esta forma, se hacen llamadas a funciones especiales que permiten realizar acciones específicas para el *hardware* de la computadora con el fin de desplegar gráficos pero todo desde un programa escrito en uno o varios lenguajes de programación de los antes mencionados. Las dos API gráficas principales, más usadas y soportadas prácticamente por cualquier fabricante de *hardware* enfocado a los gráficos por computadora son Open GL (tanto en sus versiones estándar y ES para sistemas embebidos) y Direct 3D.

Por supuesto, con el paso del tiempo, y con el surgimiento de motores gráficos más sencillos y prácticos, la tendencia actual es desarrollar el *software* gráfico siempre sobre un motor gráfico sin utilizar directamente los lenguajes y las llamadas a las API gráficas. Sin embargo es importante recordar que el motor gráfico en esencia está creando precisamente un programa con llamadas a una API gráfica solo que de manera automatizada y aunque no sea siempre visible o transparente al programador, sigue siendo el mismo principio.

En el caso de algunas empresas, estas desarrollan su propio motor gráfico y eligen la API gráfica que desean que utilice su motor. Otros motores, disponibles comercialmente como Unity o Unreal, proveen la posibilidad de crear programas ejecutables con cualquiera de las dos API gráficas e incluso para sistemas embebidos (principalmente para aplicaciones móviles en Android y iOS).

-Direct3D: es una API para gráficos por computadora soportado por Microsoft principalmente para el sistema operativo Windows. Actualmente y tras años de desarrollo constante Direct3D se encuentra en su versión 12 recién migrando de la 11. La gran mayoría de aplicaciones gráficas serias, y videojuegos, trabajan sobre esta API debido al grado de soporte que tiene y debido a que es la API por defecto en que trabaja Windows, tanto es así que Windows 10 incluye como una de sus novedades, el trabajar sobre Direct3D 12 (pasando de Direct 11 con el que trabajaban Windows 8 y 8.1). Es prácticamente obligatorio para cualquier aceleradora gráfica actual soportar Direct 3D preferentemente en su última versión.

-framerate: término en inglés que puede traducirse como “tasa de cuadros”, se refiere a la frecuencia con la que se están refrescando cuadros de imagen en un dispositivo de despliegue. Entiéndase por cuadro una imagen que corresponde al tamaño exacto en píxeles del dispositivo de despliegue. Tanto para aplicaciones serias como para videojuegos siempre se trata de lograr un valor estable de al menos sesenta cuadros por segundo (abreviado 60 fps), sin embargo esto requiere un poder de procesamiento de la

aceleradora gráfico mínimo que podría o no ser suficiente para las demandas de cada programa gráfico en particular.

-GNU: licencia GNU se refiere a una licencia para *software* y otros tipos de trabajos a los que sea aplicable la misma. Actualmente en su versión 3.0, se diferencia de otros tipos de licencias por el hecho de que se considera permisible compartir y modificar cualquier versión de un programa dado bajo una serie de reglas establecidas en la licencia. En general, será posible modificar el programa original fuente para realizar funciones diferentes a las originales o bien utilizar partes del código original que nos sean útiles para propósitos específicos. Todas las reglas pueden ser consultadas en https://www.gnu.org/licenses/gpl-3.0.en.html. **[11]**

Puede llegar a existir cierta confusión entre los términos libre y gratuito, en especial en el idioma inglés donde la palabra *free* se puede interpretar de ambas formas según el contexto. Sin embargo el *software* libre solo lo es en el sentido de las restricciones de modificación y redistribución del mismo, más no del precio del mismo. En otras palabras podremos encontrar *software* libre y gratuito, pero también existe *software* libre y con costo.

-LGNU: Traducido de sus siglas en inglés como "Licencia GNU reducida". Esta licencia es una variación de la licencia original GNU que de igual forma aplica principalmente para *software* y otros tipos de trabajos. Al ser una variación de la GNU se encuentra también en su versión 3.0 y su característica principal, es que añade algunas limitaciones adicionales. En esencia nos indica que al utilizar cualquier programa cubierto por LGNU se deberá indicar explícitamente que el programa propio utiliza dicho programa o librería, que dicho programa o librería está cubierto por la licencia GNU Lesser e incluir una copia de la licencia correspondiente con el programa propio además de la licencia GNU general. Adicionalmente, en caso de que en el programa exista un despliegue de datos de derechos de autor, se debe hacer la mención correspondiente del trabajo cubierto por LGNU utilizado. **[12]**

-*Open GL*: es una API gráfica para gráficos por computadora. Como su nombre lo indica, se trata de un gran proyecto libre que igual que Direct3D ha estado siendo desarrollado y mejorado con el paso de los años. Actualmente se encuentra en su versión 4.5 en su implementación estándar y en el caso de ES para sistemas embebidos (principalmente sistemas operativos móviles como Android y iOS) se encuentra en su versión 3.1.

Principalmente es utilizada en sistemas Linux, Unix, sistemas operativos móviles como Android, iOS y Windows Phone (en su implementación ES para sistemas embebidos). Sin embargo Windows también provee soporte para esta API gráfica, por lo que es posible ejecutar y desarrollar programas gráficos con la misma sin ningún problema. Es

prácticamente obligatorio que cualquier aceleradora gráfica soporte Open GL, de preferencia en su última versión o de lo contrario muchos programas que hacen uso de estas APIs no podrían funcionar. Un detalle importante a recalcar es que dependiendo de la versión de la API gráfica y en especial las últimas versiones (Direct3D 11, 12 y Open GL 4.4 y 4.5) requieren no solo su correspondiente paquetería de programas sino que además, las aceleradoras gráficas tienen características específicas para soportar estas APIs implementadas en *hardware*, en otras palabras si una aceleradora soporta solo Direct3D 10, será imposible que ejecute acciones que sean exclusivas de Direct 11 o 12, lo mismo para Open GL. Por supuesto, se puede lograr un soporte parcial vía *software*.

-*Pipeline* gráfico:

Se refiere a todas las operaciones básicas que se ejecutan en una aceleradora gráfica para entregar a partir de una representación 3D un cuadro 2D, mismo que tendrá el tamaño exacto en píxeles adecuado para ser mostrado en un dispositivo de despliegue. Estas operaciones se ejecutan cíclicamente para lograr obtener una cierta cantidad de cuadros por segundo. **[20]**

Las operaciones que se realizan en el *pipeline* gráfico son: **[20]**

*Preparación de datos de vértices y *renderización* de vértices

*Procesamiento de vértices

*Pos procesamiento de vértices

*Ensamblado de primitivas

**Rasterización*

*Procesamiento de fragmentos

*Procesamiento por muestra

Es importante notar que ciertas fases del *pipeline* gráfico son automáticas y en condiciones normales no pueden ser manipuladas directamente por el programador pues en muchos casos estas operaciones son de bajo nivel y se ejecutan directamente en el *hardware* de la aceleradora gráfica. Aquellas fases que son programables y manipulables por el programador, se controlan a través de lo que llamamos *shader*.

-Polígono: representación mínima posible utilizada para formar objetos 3D en aceleradoras gráficas. Si bien se denomina polígono, en realidad todas las aceleradoras gráficas transformarán cualquier polígono que deba ser *renderizado* a un grupo de

triángulos para poder procesarlo, este proceso es omitido si el objeto original ya está formado exclusivamente por triángulos. Comúnmente se utiliza la expresión “polígonos por segundo” cuando se quiere indicar el rendimiento teórico de una aceleradora gráfica.

-*Shader*: Un *shader* es un programa escrito en lenguajes especiales de programación, diseñado para ser ejecutado en un procesador gráfico. **[21]**

Su razón de ser, es la de ejecutar una de las etapas programables del *pipeline* gráfico. O como se ha hecho en los últimos años, para realizar cálculos de propósito general sobre una GPU con el fin de aprovechar las ventajas de esta sobre una CPU convencional. **[21]**

En la actualidad se consideran los siguientes tipos: **[21]**

-*Shaders* de vértices

-*Shaders* de control de teselación y evaluación

-*Shaders* de Geometría

-*Shaders* de Fragmentos

-*Shaders* de Cómputo

-TUIO: TUIO es un paquete de programas disponibles en línea bajo la licencia GNU, que en pocas palabras, proveen todas las bibliotecas, clases y métodos necesarios para interpretar entradas táctiles de diversos dispositivos de entrada (dispositivos Android, iOS, pantallas táctiles, etc) en programas de diferentes lenguajes de programación, principalmente C++, C# y Java.

Bibliografía

- **[1]** Educar para Escribir. Juan Antonio García Nuñez. Editorial Limusa. México. 2003.

  *Página 86. Capítulo 6, “Desarrollo Metodológico”. Segundo Nivel. Ejercicio 4.B.1,Trazo horizontal completo

  *Página 77. Capítulo 6, “Desarrollo Metodológico”. Segundo Nivel. Ejercicio 1.B.1, Trazo vertical completo

  *Página 106. Capítulo 6, “Desarrollo Metodológico”. Segundo Nivel. Ejercicio 12.B.1, Trazo mixto (Horizontal-vertical)

- **[2]** Reeducación Funcional tras un Ictus. José Castillo Sánchez, Isabel Jiménez Martín. Editorial Elsevier. Barcelona, España. 2015.

  *Páginas 23 a 25. Capítulo 3, “Etiología y fisiopatología del ictus”

  *Páginas 89 a 91. Capítulo 9, “Rehabilitación del ictus cerebral: evaluación, pronóstico y tratamiento”

  *Página 124. Capítulo 11, ”Estimulación magnética transcraneal como nueva estrategia terapéutica en el ictus”

Mesografía

- **[3]** Application programming interface. (2016, October 16). Wikipedia, The Free Encyclopedia. Consultado 16:08, Agosto 22, 2016, de https://en.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=744649534
- **[4]** Apps. (Sin información). Dirección General de Cómputo y de Tecnologías de Información y Comunicación-Departamento de Visualización y Realidad Virtual. Consultado 22:48, Noviembre 24, 2016 de http://www.ixtli.unam.mx/apps
- **[5]** Brain. (2017, Enero 1). Wikipedia, The Free Encyclopedia. Consultado 21:33, Enero 7, 2017, de https://en.wikipedia.org/w/index.php?title=Brain&oldid=757736777
- **[6]** Control de la motricidad fina. (2015, Mayo 2). Medline Plus. Consultado 20:00, Enero 7, 2017 de https://medlineplus.gov/spanish/ency/article/002364.htm
- **[7]** Coordinación ojo-mano. (Sin información). Lic. Rigoberto Tamayo. Consultado 16:00, Julio 18, 2016, de https://www.cognifit.com/es/habilidad-cognitiva/coordinacion-ojo-mano
- **[8]** Espasticidad. (2015, Febrero 3). Medline Plus. Consultado 21:21, Noviembre 24, 2016 de https://medlineplus.gov/spanish/ency/article/003297.htm
- **[9]** Extreme Programming: A gentle introduction. (2013, Octubre 8). Don Wells. Consultado 18:00, Julio 19, 2016, de http://www.extremeprogramming.org/
- **[10]** GLOBAL HEALTH ESTIMATES SUMMARY TABLES: PROJECTION OF DEATHS BY CAUSE, AGE AND SEX, BY WORLD BANK INCOME GROUP AND WHO REGION. (2013, Julio). Organización Mundial de la salud. Consultada 22:00, Noviembre 8, 2016, de http://www.who.int/entity/healthinfo/global_burden_disease/GHE_DthWHOReg6_Proj_2015_2030.xls?ua=1
- **[11]** GNU General Public License. (2007, Junio 29). Free Software Foundation, Inc. Consultado 21:31, Octubre 26, 2016, de https://www.gnu.org/licenses/gpl-3.0.en.html
- **[12]** GNU Lesser General Public License. (2007, Junio 29). Free Software Foundation, Inc. Consultado 17:39, Noviembre 13, 2016, de https://www.gnu.org/licenses/lgpl.html
- **[13]** Introducción a la Programación Extrema. (2002, Diciembre 30). M. en C. Aguilar Sierra Alejandro. Consultado 20:00, Julio 21, 2016, de http://www.revista.unam.mx/vol.3/num4/art39/

- **[14]** Medical Realities announces first product at London Wearable Technology Conference. (2015, Marzo 11). Edward Miller. Consultado 23:23, Noviembre 24, 2016 de http://www.medicalrealities.com/medical-realities-announce-the-virtual-surgeon/
- **[15]** Neurorehabilitación. (Sin información). Neurae. Consultado 16:00, Julio 12, 2016, de http://www.neurae.com/neurorehabilitacion/
- **[16]** Neurehabilitación. (Sin información). Santiago Ramón y Cajal. Consultado 14:00, Julio 14, 2016, de http://www.neurorehabilitacion.com/neurorehabilitacion.htm
- **[17]** Peripheral nervous system. (2016, Noviembre 28). Wikipedia, The Free Encyclopedia. Consultado 21:44, Noviembre 28, 2017, de https://en.wikipedia.org/w/index.php?title=Peripheral_nervous_system&oldid=751820575
- **[18]** Realidad Aumentada. (Sin información). Centro de difusión de Ciencia y tecnología-Instituto Politécnica Nacional. Consultado 22:00, Noviembre 24, 2016 de http://www.cedicyt.ipn.mx/RevConversus/Paginas/RealidadAumentada.aspx
- **[19]** Realidad Virtual. (Sin información).Facultat d'Informàtica de Barcelona (Universitat Politècnica de Catalunya). Consultado 22:00, Octubre 17, 2016, de http://www.fib.upc.edu/retro-informatica/avui/realitatvirtual.html
- **[20]** Rendering Pipeline Overview. (2015, Mayo 11). OpenGL.org. Consultado 03:50, Octubre 18, 2016, de http://www.opengl.org/wiki_132/index.php?title=Rendering_Pipeline_Overview&oldid=12521
- **[21]** Shader. (2015, Mayo 6). OpenGL.org. Consultado 03:47, Octubre 18, 2016, de http://www.opengl.org/wiki_132/index.php?title=Shader&oldid=12482.
- **[22]** Stroke Statistics. (Sin información).The Internet Stroke Center. Consultado 21:50, Octubre 26, 2016, de http://www.strokecenter.org/patients/about-stroke/stroke-statistics/
- **[23]** Tarjeta gráfica. (2016, Noviembre 9). Wikipedia, la enciclopedia Libre. Consultado 17:11, Noviembre 13, 2016 de https://es.wikipedia.org/w/index.php?title=Tarjeta_gr%C3%A1fica&oldid=94886367.
- **[24]** Terapia Visual como método de mejora de la coordinación ojo-mano y el tiempo de reacción visual de un tenista. (2014, Septiembre 29). Instituto de Salud Visual. Consultado 22:28, Novembre 24, 2016, de

http://institutosaludvisual.es/terapia-visual-como-metodo-de-mejora-de-la-coordinacion-ojo-mano-y-el-tiempo-de-reaccion-visual-de-un-tenista/

- **[25]** Virtual Reality in the Military. (Sin información). Virtual Reality Society. Consultado 20:00, Octubre 18, 2016, de http://www.vrs.org.uk/virtual-reality-military/
- **[26]** Nearly 1 in 6 of world's population suffer from neurological disorders – UN report. (2007, Febrero 27). Organización de las Naciones Unidas. Consultado 16:37, Agosto 2, 2016, de http://www.un.org/apps/news/story.asp?newsid=21689&cr=neurological#.V-SiWfnhCUl

Atribución de créditos:

Para el presente desarrollo de tesis, se utilizó el *plugin* “unity3d-tuio” de la empresa *Mindstorm* obtenido de la dirección https://code.google.com/archive/p/unity3d-tuio/ .

De acuerdo a la página donde fue obtenida, el *plugin* se encuentra cubierto por la licencia LGNU, por lo cual fue necesario hacer esta mención, además de incluir el *link* de donde fue obtenido, y *links* a las licencias GNU y LGNU:

-https://www.gnu.org/licenses/gpl-3.0.en.html

-https://www.gnu.org/licenses/lgpl.html

Links adicionales:

A continuación se indica el *link* del cual se obtuvo el SDK de *Kinect For Windows* y el paquete provisto por Microsoft para Unity 3D:

https://go.microsoft.com/fwlink/p/?LinkId=403899