IMPLEMENTACION DE VIDEOJUEGOS COMO HERRAMIENTA PARA EL DESARROLLO MOTOR Y COGNITIVO DE NIÑOS

Trabajo Final presentado para obtener el título de Ingeniero de Sistemas

Por
Mauricio Piccolo y Maximiliano Fabián

Director
Dr. Cristian García Bauza
Agradecimientos

A nuestras familias y amigos que fueron quienes nos dieron todo el apoyo y confianza para llegar a cumplir este objetivo. A nuestro director que nos guio y aconsejó en este proceso, a todos los profesores por ayudarnos en nuestra formación y en especial agradecimientos a la República Argentina por brindarnos la oportunidad de obtener una formación universitaria pública de calidad.
Indice General

Contenido
Agradecimientos .................................................................................................................. 1
Indice General ...................................................................................................................... 2
Resumen ............................................................................................................................... 4
Capítulo 1 .................................................................................................................................. 5
  Introducción .......................................................................................................................... 5
  Motivación ............................................................................................................................ 6
  Objetivos .............................................................................................................................. 6
  Organización del documento .............................................................................................. 7
Capítulo 2 .................................................................................................................................. 8
  Marco teórico y antecedentes ............................................................................................. 8
  Interacción hombre-máquina .............................................................................................. 8
  Dispositivos destinados a NUI ............................................................................................. 9
    Wiimote ................................................................................................................................ 10
    PS Move ............................................................................................................................ 10
    Kinect .................................................................................................................................. 11
  Videojuegos ........................................................................................................................... 13
    Videojuegos en la sociedad actual ..................................................................................... 13
    Clasificación de Videojuegos ............................................................................................ 14
    Videojuegos en el Aprendizaje ........................................................................................... 16
    Problemas vigentes de los videojuegos ............................................................................ 17
    Videojuegos y aplicaciones en la educación especial ..................................................... 17
    Kinect aplicado a educación ............................................................................................ 19
Capítulo 3 .................................................................................................................................. 22
  Desarrollo ............................................................................................................................. 22
  Tecnologías utilizadas .......................................................................................................... 22
    Microsoft Kinect SDK .................................................................................................... 22
    LibFreenect ....................................................................................................................... 23
    OpenNI .............................................................................................................................. 23
    XNA ..................................................................................................................................... 24
    Unity .................................................................................................................................... 25
    Box2d .................................................................................................................................. 25
    Cocos2D ............................................................................................................................. 26
El mundo bidimensional ................................................................. 27
Sprites ............................................................................................ 27
Motor de física - conceptos centrales .............................................. 28
Game Loop ..................................................................................... 30
Game Loop en XNA ......................................................................... 32
Diseño de la aplicación ................................................................... 37
Representación gráfica de elementos .............................................. 37
Representación física de la interacción entre elementos ................. 43
Representación de elementos específicos del juego ....................... 47
Representación de jugadores ......................................................... 49
Interacción con Kinect .................................................................... 52
Manejo de escenas ........................................................................... 55
Administrador de pantallas ........................................................... 60
Capítulo 4 ....................................................................................... 62
Instanciación ................................................................................... 62
Videojuego Círculos ........................................................................ 63
  Incorporación del jugador al videojuego ....................................... 65
  Incorporación de simulación física ............................................. 69
  Incorporación de elementos al mundo físico .............................. 72
  Validación de la lógica del videojuego ....................................... 75
  Configuración personalizada ..................................................... 78
Videojuegos adicionales ................................................................. 79
  Videojuego Fechas ....................................................................... 79
  Videojuego Seleccionador ........................................................... 83
Capítulo 5 ....................................................................................... 87
Conclusiones .................................................................................. 87
Trabajos Futuros ............................................................................. 88
Bibliografía ..................................................................................... 90
Resumen

No hay que pensar en un futuro muy lejano para darnos cuenta de que la tecnología es cada vez más natural e intuitiva. Día a día muchas personas usan tecnología táctil o de reconocimiento de voz para interactuar con teléfonos, cajeros automáticos, tiendas de autoservicio y automóviles. La curva de aprendizaje para trabajar con computadoras representa cada vez menos una barrera, gracias a que ahora hay formas más naturales de interactuar.

La tecnología comienza a comportarse como nosotros lo hacemos, la tecnología de interfaz natural de usuario (NUI, por sus siglas en inglés) es un área en la que ha invertido mucho en estos últimos años. Los frutos de dichas inversiones ahora se están plasmados en varios productos que están surgiendo en el mercado. Un producto que ha llamado mucho la atención es Kinect para Xbox 360, el cual incorpora reconocimiento facial y control por voz y gestos. El dispositivo reconoce a la persona que está enfrente, entiende su voz o el movimiento de su cuerpo. Con estas nuevas formas de interactuar, la forma de jugar videojuegos cambió y seguirá cambiando.

La tecnología natural e intuitiva como Kinect puede ser más que sólo una gran plataforma de juegos y entretenimiento; también puede generar increíbles oportunidades para resolver problemas sociales como servicios médicos, educación y mayor acceso para personas con algún tipo de limitación en sus capacidades.

En este trabajo se propone una plataforma de desarrollo de videojuegos orientados a niños con algún tipo de limitación en sus capacidades. La plataforma permite el desarrollo de aplicaciones integrando diferentes tecnologías, sin tener que dedicar tiempo y esfuerzo en integrarlas. Esto mismo permite agilizar mucho la tarea del desarrollador, que únicamente se debe ocupar en la implementación de los aspectos que se relacionan al juego en sí mismo.
Capítulo 1

Introducción

La Tecnología se define como el conjunto de conocimientos y técnicas que, aplicados de forma lógica y ordenada, permiten al ser humano modificar su entorno material o virtual para satisfacer sus necesidades. Es decir, un proceso combinado de pensamiento y acción con la finalidad de crear soluciones útiles.

Asimismo, la tecnología responde al deseo y la voluntad que tenemos las personas de transformar nuestro entorno y el mundo que nos rodea, buscando nuevas formas de satisfacer nuestros deseos, aplicándola en distintos campos en donde se desenvuelve el hombre. Particularmente en el campo educacional, la tecnología proporciona herramientas eficientes a docentes y alumnos para ofrecer entornos de aprendizaje que aseguren condiciones propicias, equitativas y que, a su vez, permitan individualizar la enseñanza a efectos de responder a la amplia gama de necesidades de aprendizaje de sus alumnos.

Las propuestas educativas basadas en multimedios se fundamentan en un aprendizaje constructivista en el que los alumnos exploran relaciones y emplean experiencias visuales, auditivas y cinestésicas para adquirir nuevas destrezas y conocimientos en base a su aprendizaje previo. Los videojuegos que recrean el mundo real o crean nuevos mundos, les proponen a los niños y jóvenes, desafíos y problemas a resolver a partir de metodologías de motivación que los involucra y compromete con la situación planteada.

Los ambientes virtuales llevan a los usuarios a realizar una actividad cognitiva caracterizada por la multitarea, evaluación de alternativas y toma de decisiones, los rápidos y constantes cambios de atención entre objeto y contexto, transgresión de lo real y el intercambio con pares. Los videojuegos aplicados en ámbitos de atención a la discapacidad pueden tomar varias formas: como ocio o entretenimiento, como recurso educativo, y como herramienta rehabilitadora. En cualquiera de los casos es necesario la participación activamente en el videojuego (que sean accesibles y adaptados) y así obtener beneficios en el desarrollo personal, se tenga o no una discapacidad. La posibilidad de manipular y transformar objetos en el espacio de la interfaz comprometiendo un conjunto diferente de competencias cognitivas, les otorga a las actividades educativas cierto grado de fortalecimiento pedagógico. Algunas investigaciones resaltan efectos beneficiosos de utilizar videojuegos como el fomento de la creatividad, propiciar el entrenamiento y la mejora de habilidades, ya sea con fines terapéuticos o didácticos.

Considerando las características de los destinatarios, los aspectos pedagógicos y el diseño de herramientas informáticas antes mencionados es que se espera desarrollar una
aplicación de tecnología asistencial que pueda ser utilizada para aumentar, mantener o mejorar las capacidades funcionales de personas con discapacidad motora o cognitiva y que colabore para la inclusión social de niños y jóvenes con discapacidad. En particular, se propone implementar un framework que permite desarrollar diversos videojuegos aplicados al área.

Por otro lado, con la evolución del hardware se ha logrado que las interacciones de los usuarios con las computadoras hayan mejorado rotundamente, generando nuevas formas de comunicación y relación con los dispositivos.

En este aspecto, puede considerarse un hito el dispositivo Kinect de Microsoft con el que se permite interactuar con una computadora a través de gestos y movimientos realizados con el cuerpo. Kinect es capaz de reconocer el movimiento del usuario mediante su hardware interno, de manera tal de obtener información de cada parte del cuerpo humano y mapearlo de manera automática a la pantalla de la PC. Este dispositivo fue inicialmente dirigido a consolas de videojuegos, más precisamente XBOX 360, pero es posible conectarlo a una PC y poder interactuar con el Sistema Operativo. Teniendo en cuenta este último aspecto, se han diseñado una gran cantidad de programas utilizando dicha tecnología, pero aun teniendo un kit de desarrollo estable todavía no se han desarrollado muchas plataformas de videojuegos y menos aún destinados a juegos de niños que posean algún tipo de discapacidad.

Motivación

A diferencia de otros frameworks de videojuegos, nuestra propuesta brinda la posibilidad de crear videojuegos que están orientados a un segmento de mercado definido por los usuarios que poseen algún tipo de limitación en sus capacidades. A través de la plataforma propuesta se podrá crear videojuegos con integración de Kinect de una manera más sencilla y dinámica, con la que pretendemos que se disminuya en gran medida el tiempo empleado a la hora de desarrollar videojuegos con este dispositivo destinado a fines terapéuticos- educacionales.

Objetivos

En este trabajo se propone diseñar e implementar una plataforma de videojuegos que utilicen el dispositivo Kinect que puedan ser utilizados para mejorar las terapias de niños que poseen algún tipo de discapacidad. Se pretende que dicha plataforma provea una forma rápida y sencilla para implementar videojuegos que sean requeridos por profesionales que busquen realizar actividades para incentivar a los niños para que puedan incorporar nuevos movimientos, gestos o capacidades.
Dentro del trabajo se proveerá un ámbito en donde se podrá elegir los distintos videojuegos desarrollados en la plataforma para que los profesionales puedan utilizar cada uno de los mismos.

**Organización del documento**

En este documento se describen todos los procesos de análisis y toma de decisiones que se llevaron a cabo durante el desarrollo de la plataforma, ya mencionada previamente dentro de la introducción y motivación de nuestro trabajo.

Dentro del capítulo 2 se definen y explican conceptos esenciales relacionados a los videojuegos, dispositivos de interfaz natural y las posibles aplicaciones en el ámbito educacional de los mismos utilizando las nuevas tecnologías emergentes en el mercado.

En del capítulo 3 se puede ir recorriendo el proceso por el cual la arquitectura de la plataforma se va adaptando a las necesidades de desarrollo de la plataforma para llegar a cumplir con los objetivos propuestos. En dicho capítulo se van narrando lo diversos cambios que se introducen en la arquitectura de la plataforma, incluyendo diagramas de clases para facilitar la comprensión de la misma.

El capítulo 4 muestra cómo se utilizó la plataforma para instanciar una serie de videojuegos desarrollados a medida para el instituto ATAD. En éste capítulo se muestra más en profundidad la estructura de un juego desarrollado con la plataforma, mediante diagramas de clases y código fuente de cada una de las instancias realizadas.

Dentro del último capítulo se realiza una breve conclusión enumerando las ventajas y desventajas de la plataforma, en base a la experiencia obtenida a lo largo del desarrollo y aplicación de la misma. Además, se enumeran algunos de los posibles trabajos a realizar en un futuro cercano en busca de mejorar la plataforma.
Capítulo 2

Marco teórico y antecedentes

En este capítulo se introducen algunos conceptos relacionados al desarrollo de videojuegos con Kinect, también se hace énfasis en la interacción Hombre-Computadora, especialmente en dispositivos referidos al seguimiento de movimientos humanos y su interpretación. Adicionalmente, se describen conceptos asociados a la creación de videojuegos y su clasificación, analizando su evolución y la implicancia en la educación en conjunto con el dispositivo Kinect.

Interacción hombre-máquina

La interacción hombre-máquina o en inglés Human Computer Interaction (HCI) fue evolucionando a través del tiempo y más aceleradamente en estos últimos años [1]. A lo largo de los años se fueron desarrollando diversas formas de interacción. Dentro de las primeras, se encuentra la interacción mediante el uso de líneas de comando a través de una consola de ejecución, en inglés Command Line Interaction (CLI). Las instrucciones se envían a través de palabras que se interpretan como acciones a ejecutar, las mismas son introducidas por un teclado y el resultado a la interacción del comando es mostrado en una pantalla o monitor. Esta modalidad de interacción es todavía muy usada en algunos ámbitos muy específicos. Como aspecto positivo, se puede decir que es una forma de interacción muy potente en su versatilidad de uso para ejecutar instrucciones complejas en una computadora. La principal desventaja de esta forma de interacción es el grado de especialización que debe tener el usuario para poder recordar y armar instrucciones secuenciales o anidadas para poder interactuar con la computadora.

El siguiente paso en la evolución en la interacción hombre-máquina fue el del uso de entornos gráficos que permitieron el uso más amigable de una computadora, conocidos como interfaz gráfica de usuario o en inglés Graphical User Interface (GUI). El entorno gráfico es mucho más amigable comparado a aquellos ambientes basados en una consola de ejecución de línea de comandos. Esta forma de interacción tiene la principal desventaja en limitarse a definir un número pequeño de comandos ya que depende del área de interacción gráfica para ofrecerlos. Aun así, es una forma de interacción muy usada y amigable para el desarrollo de software, sobre todo si se requiere que el usuario final de software no requiere de conocimientos especializados sobre manejo de comandos.
La interfaz natural de usuario (NUI) de sus siglas en inglés (Natural User Interface) es una evolución de la interfaz gráfica de usuario (GUI). Surge como un mecanismo de interacción hombre-máquina que permite establecer una comunicación con la computadora a través de periféricos que pueden recibir instrucciones e información. También hace uso de periféricos de salida para mostrar la información o el resultado del sistema a la interacción hombre-máquina. La NUI se enfoca en hacer uso de una comunicación de manera natural al ser humano, haciendo uso de captar información en tiempo real logrando una interacción corporal de manera directa, sin utilizar un periférico de intermediario para la entrada de información. La utilización de NUI se ha expandido en los últimos años debido a su integración con el mercado de las consolas de videojuegos, tales como: Kinect de Microsoft, para la consola XBOX 360, WiiMote en el caso de Nintendo y también PSMove de Sony.

![Figura 1 – Evolución de la Interacción Humano Computadora (HCI).](image)

**Dispositivos destinados a NUI**

En los últimos años fueron surgiendo distintos dispositivos desarrollados y comercializados por empresas líderes del mercado de tecnologías como Sony, Microsoft, Nintendo, entre otras. En este punto se mencionan algunos de los dispositivos que hacen posible la interfaz natural en algunas consolas de videojuegos.
Wiimote

La empresa Nintendo fue la primera en comercializar un dispositivo por medio del cual se incorporan los movimientos del jugador a los juegos de su consola Wii. El Wiimote [2] tiene la capacidad de detectar la aceleración a lo largo de tres ejes mediante la utilización de un acelerómetro ADXL330. También cuenta con un sensor óptico PixArt que permite determinar el lugar al que el Wiimote está apuntando y la distancia a la que se encuentra el jugador. Esto es posible gracias a la barra de leds que provee la consola. La misma permite al sensor óptico captar los rayos infrarrojos que son emitidos por la misma y mediante triangulación se calcula la distancia que existe entre el usuario y la barra de leds. El Wiimote al ser manipulado con una mano y mientras el usuario realiza movimientos, los sensores recogen datos de aceleración en los ejes X, Y, Z y mediante la aplicación de algoritmos se determina el movimiento realizado por el usuario.

PS Move

Sony fue la primera compañía que intentó duplicar la tecnología de Nintendo. Incluso lo ha mejorado con el denominado PlayStation Move [3] para PlayStation 3. Este sistema comparte varias tecnologías con el Wiimote, aporta varias novedades que hacen que la captación del movimiento del jugador sea mucho más fiel. Al igual que el Wiimote, Move también dispone de un acelerómetro de tres ejes, que resulta en teoría mucho más preciso a la hora de captar los movimientos y rotaciones del jugador. Adicionalmente posee un sensor magnético terrestre diseñado específicamente para captar la posición espacial del jugador, es decir, la posición en la habitación en la que nos encontremos. También posee
una cámara, llamada PlayStation Eye, que es la que realiza el reconocimiento de objetos, y en conjunción con lo anteriormente comentado, traslada los movimientos a la pantalla de juego.

![Figura 4 – Sony PS Move](image)

Kinect

La última compañía que presentó otro sistema de detección de movimiento fue Microsoft, que sorprendió a todo el mercado con su dispositivo Kinect, dado que elimina por completo la necesidad de uso de un mando. Por el contrario, todas las campañas de publicidad de Microsoft aseguran que el jugador es el propio mando, ya que Kinect escanea y capta el cuerpo del jugador y su movimiento, con el fin de trasladar estos movimientos a la pantalla. Además, Kinect cuenta con un sistema de reconocimiento de voz que permite que el jugador controle algunas de las funciones de la consola y algunas de las acciones de los juegos, mediante comandos de voz.

Kinect se basa en una tecnología propietaria de la compañía PrimeSense, que hace uso de una cámara RGB, un sensor de profundidad, un micrófono de múltiples matrices y un procesador personalizado que proporcionan captura de movimiento de todo el cuerpo en 3D, reconocimiento facial y reconocimiento de voz. El sensor de profundidad se basa en un proyector de infrarrojos combinado con un sensor CMOS monocromo que permite a Kinect ver la habitación en 3D en cualquier condición de luz ambiental. Además, cuenta con un sensor de inclinación sumamente avanzado.
La instalación de Kinect es muy sencilla, el sensor debe estar colocado de 60cm a 2m sobre el suelo y centrado con la pantalla. Hay que asegurarse de colocar Kinect en una superficie firme y segura, cerca del borde. También hay que asegurarse de que ningún rayo de sol le dé directamente al sensor o al jugador. Por otro lado, Kinect necesitará ver al jugador de cuerpo entero, esto quiere decir: de pies a cabeza.

![Figura 5 - Elementos asociados al dispositivo Kinect](image)

Mediante la cámara con sensor de profundidad realiza un mapeo de 20 puntos de diferentes partes del cuerpo del jugador. Las coordenadas obtenidas de estos puntos van cambiando cuando el jugador hace algún tipo de movimiento.

![Figura 6 - Conformación de los puntos del esqueleto obtenido con Kinect](image)

![Figura 7 - Mapeo corporal completo (izq) y parcial (der)](image)
Las primeras versiones del SDK de Kinect únicamente permitían el mapeo del cuerpo del jugador parado, pero luego en versiones posteriores se incluyó en el SDK el modo de jugador sentado, ofreciendo menor cantidad de puntos a mapear.

Como se mencionó antes, Kinect estaba destinado a ser usado con la consola Xbox. Dada la potencialidad y lo novedoso que la comunidad lo encontró al dispositivo, en poco tiempo se desarrollaron SDKs no oficiales, creados por las mismas comunidades, con el fin de utilizarlo como periférico de entrada en computadoras. Esto hizo crecer en gran número las ventas de Kinect e incluso los proyectos desarrollados por esas comunidades cada vez eran más diversos y prometedores. Microsoft no dejó pasar la oportunidad y lanzó un SDK oficial para todos aquellos desarrolladores que quisieran usar el dispositivo en la computadora. Cada cierto tiempo fue incorporando mejoras en las nuevas versiones de su SDK oficial. Además de esto, Microsoft aprovechó el impulso de todos los proyectos de las comunidades y fomentó el uso de Kinect en distintas áreas como:

- Medicina
- Educación
- Seguridad
- Robótica
- Control autónomo de vehículos
- Marketing

Videojuegos

Un videojuego es un programa creado expresamente para divertir, basado en la interacción entre una persona y una máquina donde se ejecuta el videojuego. Los mismos recrean entornos virtuales en los cuales el jugador puede controlar a un personaje o cualquier otro elemento de dicho entorno, para conseguir uno o varios objetivos por medio de unas reglas determinadas.

Videojuegos en la sociedad actual

Desde la primera patente de videojuego en 1948 de Thomas T. Goldsmith Jr. y Estle Ray Mann, pasando por el conocidísimo Tennis o “Pong” de Higinbotham en 1958, los videojuegos se han ido introduciendo en la sociedad siendo una parte importante de ella. Actualmente es la industria que más factura superando a la música y el cine.
videojuegos se han convertido en el tipo de juego preferido, transformándose en un mecanismo de gran influencia a nivel cultural y social en niños, adolescentes e incluso adultos.

Los videojuegos deben tener algunas características para lograr atraer a los jugadores:

- Existencia de un héroe o protagonista con el cual el jugador debe identificarse sin provocarle rechazo.
- Antagonistas o retos que proponen nuevos desafíos a superar.
- Entregar una realimentación constantemente, sobre todo si los avances son significativos.
- Crear un efecto inmersivo que permita al jugador insertarse en la mecánica del juego. Lo que se ha visto como algo negativo al ser considerado, en muchos casos, un factor "adictivo".

Tradicionalmente los videojuegos se han visto como un mecanismo de ocio y entretenimiento sólo válido para los recreos de la escuela, dejando de lado sus beneficios como herramienta de aprendizaje gracias a sus efectos de estimulación, concentración y entretenimiento.

Clasificación de Videojuegos

Existe una gran variedad de videojuegos, se pueden clasificarlos según la cantidad de jugadores que participan y también es posible clasificarlos según su naturaleza. A continuación, se detalla brevemente cada uno de los tipos de videojuegos a partir de las clasificaciones mencionadas.

Según el número de jugadores

- **Videojuegos Individuales**: Un juego individual es el que recoge un solo jugador. Este se enfrenta a la Inteligencia Artificial del juego para conseguir una serie de objetivos dentro de la trama.
- **Videojuegos Competitivos**: Un juego competitivo es aquel juego donde un jugador juega para conseguir su propio éxito personal o meta. Este objetivo final puede ser totalmente opuesto al de otros jugadores, por lo que generalmente al ganar un jugador el resto de jugadores pierden.
- **Videojuegos Colaborativos**: En el otro lado de la balanza tenemos los juegos colaborativos donde el éxito individual se transforma en que el grupo consiga la meta deseada, es aquí donde aparece el concepto “equipo”. Resumiendo, un individuo
gana cuando gana el equipo y con ello todos sus miembros. La meta es común, global y única para todos los individuos del juego.

- **Videojuegos Cooperativos**: Basándose en las ideas de los puntos anteriores aparecen los juegos cooperativos, donde cada uno de los individuos puede tener sus propios objetivos individuales, aunque se agrupan en equipos para buscar un beneficio conjunto para la consecución de sus propias metas, que pueden obtener gracias a los demás. Es decir, el equipo no es primordial para conseguir la meta del jugador, sino algo circunstancial. Se pueden tener metas a nivel de equipo, pero deberán ser algo secundario.

**Según su naturaleza**

- **Acción**: Son juegos donde la acción y el movimiento, combates y batallas son los factores predominantes. Requieren mucha rapidez en la interacción del usuario con el juego.

- **Aventuras**: Estos juegos describen experiencias largas. Pueden asemejarse a una “película interactiva” donde el jugador se introduce en el mundo del juego como protagonista del mismo.

- **RPG**: El videojuego de rol, comúnmente conocido como RPG, es un tipo de videojuego que usa elementos del juego de rol tradicional (desempeñar un determinado rol o personalidad concreta). El personaje debe mejorar sus habilidades interactuando con el entorno y el argumento de la historia. Algunos elementos fuertemente asociados a los videojuegos de rol, como el desarrollo estadístico de personajes, han sido adaptados ampliamente a otros géneros de videojuegos. El objetivo principal de estos juegos es que el jugador viva una historia.

- **Estrategia**: Provocan la necesidad de crear esquemas, planear y dirigir operaciones, manejar recursos y desarrollar planificaciones a largo plazo para alcanzar objetivos concretos.

- **Simulación**: Juegos basados en la reproducción, generalmente de forma realista, del funcionamiento de alguna actividad.

- **Educativos**: Juegos cuyo objetivo es transmitir al jugador algún tipo de conocimiento. Su mecánica puede abarcar cualquiera de los otros géneros. Habitualmente son dirigidos para niños de menos de 10 años, aunque hay excepciones.

- **Carreras**: Estos juegos se caracterizan por desarrollar carreras de vehículos (motos, coches, etc.). El reto es que el jugador demuestre su pericia al volante de su vehículo y no cómo de realista es ésta, si no podrían considerarse de simulación.

- **Lucha**: Estos juegos retan al jugador a ir ganando combates de lucha para conseguir un campeonato o un premio. La mayor dificultad que presentan este tipo de juegos
es la combinación de teclas para realizar los movimientos especiales característicos de cada luchador.

- **Puzzles**: Son los juegos donde la motivación reside en resolver preguntas o problemas planteados. Estos juegos prueban la habilidad y la destreza del jugador.
- **Musicales**: Su desarrollo gira en torno a la música ya sean de tipo “karaoke” o en los que se debe bailar.
- **Deportes**: Se consideran juegos deportivos aquellos que simulan los deportes tradicionales como el fútbol, baloncesto, golf, etc.
- **Plataformas**: Juegos en los que el protagonista ha de avanzar a través de un mapa con distintas plataformas u objetos situados en diferentes alturas para llegar al objetivo.
- **Disparo**: También conocidos como “Shooters” o “Shoot'em up”. En estos juegos el protagonista ha de abrirse camino a base de disparos.
- **MMOG**: Se les llama también juegos masivos multijugador online. Se caracterizan porque los jugadores pueden interactuar con otros de forma competitiva o colaborativa. Los jugadores pueden agruparse en grupos llamados “clanes” con otros jugadores que posean rasgos en común, y complementarse entre ellos gracias a sus propias habilidades para poder “vencer” a otros clanes en determinados retos.

**Videojuegos en el Aprendizaje**

La incorporación de los videojuegos en el proceso de aprendizaje y con ello la incorporación de contenidos didácticos dentro de los programas de computadoras, ha ayudado a dar una evolución a los recursos didácticos con los que cuentan los docentes a la hora de realizar su profesión. Los videojuegos se han convertido en un medio atractivo para los alumnos y que fácilmente pueden ser utilizados en contextos diferentes dentro del aula. Existen numerosos estudios que muestran sobre los beneficios de los videojuegos como excelentes herramientas educativas, de los que podemos extraer las siguientes conclusiones [4]:

- **Éxito escolar**: Los alumnos que utilizaron videojuegos incrementaron notablemente su capacidad de comprensión lectora.
- **Habilidades cognitivas**: Los videojuegos proponen ambientes de aprendizaje basados en el descubrimiento y en la creatividad.
- **Motivación**: Los videojuegos suponen un mecanismo de estímulo para los niños, lo que facilita el proceso de aprendizaje y aumenta considerablemente la asistencia a clase.
- **Atención y Concentración**: Los juegos incrementan la atención del alumnado y su concentración a la hora de resolver un problema concreto debido a su naturaleza lúdica.

Problemas vigentes de los videojuegos

Ante las virtudes mostradas anteriormente, también se pueden encontrar una serie de problemas que surgen por el uso excesivo de los videojuegos, debido en su mayoría a un uso poco controlado. Estos puntos negativos son:

- **Agresividad**: La mayoría de los videojuegos llevan una violencia explícita en sus diálogos y acciones físicas. Se ha comprobado que la conducta agresiva dentro de los juegos se ve reflejada en el jugador, y más a edades tempranas [5].
- **Prejuicio de Género**: Los videojuegos reflejan diferencias de género a favor de la masculinidad, y esto provoca la absorción de estereotipos erróneos en la vida real.
- **Efectos inmersivos**: Algunos juegos ofrecen un efecto “hipnótico” sobre el jugador que llega a producirle un “autismo electrónico”, provocando un aislamiento de la realidad, lo que puede ser un problema en videojuegos comerciales, se puede tornar en una ventaja en el campo educativo, siempre y cuando se tenga un control sobre el juego y el jugador [6].

Es importante que a la hora de desarrollar videojuegos educativos se consideren estos problemas y se tome en cuenta que se debe direccionar el desarrollo de los videojuegos tanto a la naturaleza educativa, como a la edad del futuro jugador.

Videojuegos y aplicaciones en la educación especial

Investigando sobre el uso de videojuegos, particularmente aplicados a educación especial, en algunos casos pueden surgir ciertos problemas:

- ✓ No siempre los videojuegos educativos están orientados para personas con algún tipo de necesidad especial. Esto hace que pocos videojuegos con contenido educacional puedan ser utilizados en ese contexto.
- ✓ De los videojuegos disponibles en el mercado destinados a la educación especial, estos a veces se centran más en unidades didácticas perdiendo la esencia del juego.
- ✓ A Veces los dispositivos donde se implementan producen poco interés para el niño.

Se debe reflexionar en cómo todas las ventajas demostradas en el campo educativo pueden ser trasladadas a la educación especial. Para ello se debe proponer nuevos mecanismos de diseño e interacción que ayuden a superar las deficiencias cognitivas y permitan eliminar la “brecha digital” existente dentro de este campo, ayudando al proceso de integración social.
A la hora de desarrollar un videojuego orientado a la educación especial no se deben perder de vista que se va a desarrollar un juego, y que éste llevará incluidos los contenidos educativos. Por ello se debe tener en cuenta una serie de puntos:

- **Identificar para qué tipo de usuario va orientado el juego.** Sobretodo identificar las limitaciones cognitivas que presenta a la hora de jugar y de relacionarse con el entorno. Esto es fundamental para desarrollar los mecanismos de interacción multimodal y elegir los estímulos adecuados que permitan al jugador poder disfrutar del juego sin estar cohibido debido a sus limitaciones.

- **Estructurar y adaptar la unidad didáctica a la naturaleza del jugador y del juego.** Es importante llegar a un equilibrio en qué se quiere enseñar y cómo se debe enseñar.

- **El contenido educativo debe ser introducido de manera oculta, camuflado dentro la estructura del juego, teniendo éste sus propios objetivos de aprendizaje.** El objetivo es que el alumno-jugador, solo debe preocuparse de jugar, y de resolver los problemas planteados en el juego. El aprendizaje debería llegar de una manera implícita.

- **Se deben valorar los aspectos positivos que ofrece el juego, sin olvidar los aspectos negativos que pueden crear vicios o conductas inapropiadas en el niño.**

**Conectar igualdad**

Conectar Igualdad fue creado en abril de 2010 a través del Decreto Nº 459/10 de la Presidenta de la Nación, Cristina Fernández de Kirchner. Este Programa tiene el objetivo de entregar una netbook a todos los estudiantes y docentes de las escuelas públicas secundarias, de educación especial, y de los institutos de formación docente. Se propone, además, capacitar a los docentes en el uso de esta herramienta, y elaborar propuestas educativas que favorezcan su incorporación en los procesos de enseñanza y aprendizaje. Dentro del programa Conectar igualdad, existe una modalidad de escritorio que provee herramientas y aplicaciones destinadas a educación especial. Entre el software que se provee en esta modalidad se puede encontrar GCompris, un conjunto de aplicaciones destinadas a esa modalidad educacional.

**GCompris**

Es un conjunto completo de programas educacionales multiplataforma para niños de 2 a 10 años de edad, y forma parte del proyecto GNU. Incluye más de 140 actividades amenas que ayudan a los niños pequeños a adquirir habilidades como el reconocimiento de letras y números, el uso del ratón y del teclado, nociones básicas de álgebra, leer la hora en relojes analógicos, dibujar vectores, aprender idiomas mediante juegos, y mucho más. El programa
también incluye un simple procesador de texto para crear documentos con un estilo de texto básico. Otra característica es la herramienta de animación, que puede usarse de manera simple y eficaz como método de respaldo a la narración de cuentos, mejorando así la comunicación con los niños y ayudándolos a expresarse por sí mismos. El programa y todas las actividades están disponibles en muchos idiomas diferentes.

Un ejemplo del uso innovador de GCompris es el trabajo hecho por el Proyecto Insight [7] de Kerala, en la India. El proyecto tiene como objetivo llevar los beneficios del software libre a personas residentes del estado que poseen algún tipo de limitación en sus habilidades. Los responsables del proyecto comprobaron que GCompris es una poderosa herramienta para mejorar las habilidades motrices y comunicativas de personas con dificultades, como por ejemplo niños con autismo. En dicho proyecto se está usando y adaptando el programa a las necesidades más específicas de la comunidad destinataria. El equipo de Insight puso en práctica un módulo de formación para los docentes, con instrucciones sobre el uso de las diferentes actividades de GCompris cuando se trabaja con niños con necesidades especiales. El objetivo es ayudar a los niños a adquirir habilidades tales como la coordinación óculo-manual, la memoria, la comprensión oral, como así también mejorar la capacidad de concentración del niño. La herramienta está dirigida a diferentes niveles de educación, desde preescolar hasta primaria. GCompris también se usa para ayudar a niños mayores a entender la relación entre los números, aprender operaciones básicas de aritmética, ejercitar el razonamiento analítico y para mejorar la capacidad de comunicación, además de deletrear, leer y escribir.

Kinect aplicado a educación

Hui-Mei Hsu [8] explora el potencial de Kinect en la Educación como una tecnología interactiva y discute cómo este dispositivo puede facilitar y mejorar la enseñanza y el aprendizaje. Kinect utiliza una tecnología basada en gestos, de esta forma puede apoyar las prácticas pedagógicas kinestésicas para beneficiar a los estudiantes fortaleciendo su inteligencia corporal-kinestésica. Además, menciona que el dispositivo Kinect fomenta la participación en clase y facilita al personal docente interactuar con el material multimedia para crear momentos de interacción o debate, lo cual promueve el aprendizaje e impulsa la motivación de los estudiantes. Kinect es una herramienta de enseñanza flexible pues permite la interacción con gestos sin utilizar el teclado, es versátil porque soporta el desarrollo de varias actividades en la enseñanza, se puede adaptar a múltiples usuarios y mantiene a los estudiantes comprometidos en clase. Por otro lado Kiili [9] introduce el concepto exerbraining, entrenamiento combinado de cuerpo-cerebro, y discute los
escenarios de implementación realizando un experimento con 53 estudiantes de tercer grado de primaria en donde concluyeron que los juegos exerbraining ofrecen nuevas posibilidades para las escuelas para aumentar la actividad física, ya que pueden ser utilizados como una solución alternativa de aprendizaje que se puede aplicar en el día a día en las escuelas sin interferir con los objetivos del plan de curricular. En general, la incorporación de los juegos exerbraining en escuelas y hogares podría promover un estilo de vida más saludable para los estudiantes y el combate contra la crisis de la obesidad infantil emergente. Con respecto a los escenarios de implementación, concluyeron que los juegos exerbraining podrían ser utilizados en las escuelas de varias maneras, por ejemplo, en las clases de educación física, aulas, pasillos y en talleres después de la escuela. Además, los estudiantes disfrutaron mucho los juegos exerbraining lo cual otorga una efectiva solución para el aprendizaje dentro de las escuelas. Así mismo el uso de los movimientos corporales mejoran las tareas de memoria según las pruebas realizadas por Kuo-Jen Chao [10] haciendo uso del Kinect, lo cual provee diferentes canales de aprendizaje para los estudiantes. El uso del Kinect dentro de la educación como una tecnología emergente crea nuevas experiencias para el proceso de aprendizaje, yendo más allá del uso de los movimientos corporales a la mejora del desempeño de la memoria. Del mismo modo, el uso de Kinect aplicado en el aprendizaje de las matemáticas, ha sido de gran apoyo para los estudiantes como lo demuestra Néstor Adrián Rodríguez Ayala [11], donde se desarrolló un software para reproducir gráficos y patrones con el Kinect. El equipo de trabajo pudo recoger datos en relación con los movimientos corporales y el proceso de aprendizaje. El software fue diseñado con las herramientas de Visual Studio y el SDK para Kinect.

Las interfaces naturales como un sistema de aprendizaje, mejoran la enseñanza de Matemática en la educación secundaria, como lo demuestra Martha Alicia Magaña Echevarría [12], que con el apoyo del dispositivo Kinect lograron mejores resultados en el proceso de aprendizaje y la adquisición del conocimiento matemático. Además, el estudiante logra una mejor percepción de lo que está aprendiendo y se muestra más interesado en el curso.

Los videojuegos basados en los ejercicios mejoran el nivel académico y salud para el bienestar de los alumnos según lo demuestra Zan Gao [13] en niños de 3 a 5 años, donde adquieren mejor destreza en habilidades matemáticas y de lectura en sus aulas, la hipótesis es que disminuyen el déficit de atención, aparte de promover una vida activa saludable ayudando a los problemas de obesidad. Chi-Hung Chuang [14] demuestran que las personas se divierten mucho aprendiendo en escenarios interactivos donde utilizan el Kinect y además despiertan la percepción de competencia entre los estudiantes en el proceso de
aprendizaje. Esta actitud positiva permite que los profesores puedan expandir más los conceptos a enseñar en sus materias. Asimismo, un estudio sobre la alfabetización digital fue elaborado por Bruce D. Homer [15], donde se hizo uso del dispositivo Kinect como un medio de interacción lo que trajo muchos beneficios en la compresión de las historias relatadas, a pesar de ser un juego con actividades psicomotrices no involucró ninguna distracción, sino que dio mayores aportes hacia los niños e interés por seguir el juego y aprender a la vez.

Dadas las ventajas mencionadas en secciones anteriores sobre la aplicación de los videojuegos a la educación especial y sumado a todas las nuevas ventajas que trae aparejada la utilización de las nuevas formas de interacción. Como se mencionó anteriormente nuestro objetivo fundamental es desarrollar una plataforma que permita integrar diferentes tecnologías y facilitar al programador el desarrollo de videojuegos educacionales destinados a la educación especial de niños. En el siguiente capítulo se comenta en profundidad el diseño y desarrollo de la plataforma.
Capítulo 3

Desarrollo

El problema principal surge de la necesidad de implementar videojuegos interactivos cuyo contenido sea atractivo y simple para el usuario final. De esta manera se requiere representar cualquier tipo de objeto en un espacio bidimensional, con diferentes características; tales como tamaño, color, forma, etc.

En el diseño de la plataforma se encuentran varios paquetes que permiten el desarrollo de un juego 2D desde cero, incorporando diferentes tecnologías. Dentro de ellas se encuentran XNA Game Studio, Box 2D y Kinect SDK.

En los distintos paquetes podemos encontrar las funcionalidades que permiten la interacción del usuario final con el hardware de Kinect, representar objetos 2D dentro del juego y la incorporación de física para mejorar la interacción de los objetos representados con el usuario.

Tecnologías utilizadas

A continuación, se describen posibles alternativas para el desarrollo de la plataforma y la selección de aquellas que se eligieron por sobre las demás alternativas.

Microsoft Kinect SDK

El kit de desarrollo de Microsoft fue pensado para permitir a los desarrolladores implementar aplicaciones para Windows, basados en múltiples lenguajes de programación como C++/CLI, C#, y Visual Basic .NET.

Una de las características fundamentales del SDK es el reconocimiento del cuerpo humano. Básicamente, los datos son procesados en crudo desde el sensor permitiendo proporcionar información como el seguimiento del esqueleto (representación del cuerpo humano), con un máximo de seis personas en escena, y el reconocimiento de palabras a partir de datos de audio para un determinado idioma.

El SDK permite también captar la voz y los gestos (de seguimiento de la cara y los pequeños movimientos de todo el cuerpo). El sensor tiene cuatro micrófonos que permiten a nuestra aplicación responder a la entrada verbal. Permite elegir las palabras o frases específicas para escuchar y utilizar como disparadores, para luego traducirse a comandos.
LibFreenect

Es un kit de desarrollo open source creado por una comunidad llamada OpenKinect. Esta es una comunidad abierta de personas, aproximadamente 2000, interesadas en permitir el uso de Kinect en computadoras con sistemas operativos Linux, Os X y Windows. La librería permite a los programadores desarrollar aplicaciones utilizando las funcionalidades provistas por el hardware de Kinect. La misma está disponible para realizar desarrollos en diferentes lenguajes, tales como C, C++, C#, Java, Python y Ruby.

OpenNI

Open Natural Interaction es una organización sin ánimo de lucro impulsada por la industria. La organización fue creada en noviembre de 2010. Uno de los miembros principales era PrimeSense, la empresa que creó la tecnología utilizada en la Kinect. En diciembre de 2010, PrimeSense liberó sus propios drivers de código abierto. El SDK de OpenNI al igual que otros permiten desarrollar aplicaciones usando las funcionalidades que provee el hardware de Kinect. Es posible desarrollar aplicaciones para sistemas Windows, Linux y Mac OS X en lenguajes C++, C#, Java y Python.

En la siguiente tabla comparativa se muestran un resumen de las principales características de los tres kits de desarrollo.
<table>
<thead>
<tr>
<th>Soporte multiplataforma (Linux, MAC OS X, Windows)</th>
<th>Si</th>
<th>No</th>
<th>Si</th>
</tr>
</thead>
<tbody>
<tr>
<td>Licencia</td>
<td>Apache v2/GPL v2.</td>
<td>La licencia comercial autoriza el desarrollo y distribución de aplicaciones comerciales de Windows para clientes finales.</td>
<td>Apache 2.0</td>
</tr>
<tr>
<td>Compatibilidad XNA</td>
<td>No</td>
<td>Si</td>
<td>Si</td>
</tr>
<tr>
<td>Compatibilidad Unity</td>
<td>Si</td>
<td>Si</td>
<td>Si</td>
</tr>
</tbody>
</table>

Luego de realizar la comparación entre las alternativas que teníamos disponibles, priorizamos algunos aspectos por sobre otros para decidirnos por el SDK oficial que provee Microsoft.

- Microsoft provee una mejora continua en cuanto a funcionalidades, provee nuevas versiones estables del SDK cada cierto tiempo.
- Documentación técnica muy completa y ejemplos funcionales muy útiles.
- Compatibilidad entre hardware y software inmejorable.

**XNA**

XNA es una plataforma de desarrollo de videojuegos 2D y 3D creada por Microsoft utilizando DirectX y .NET, en la cual se ofrece cierta funcionalidad ya integrada. Esto mismo
permite al desarrollador centrarse directamente en el desarrollo y diseño del juego implementando los métodos estrictamente necesarios para que el juego funcione.

**Unity**

Es un motor de videojuegos que permite desarrollar juegos 2D y 3D creado por Unity Technologies en múltiples plataformas. El motor gráfico utiliza Direct3D en Windows, OpenGL en Mac y Linux, OpenGL ES en Android y iOS, e interfaces propietarias en Wii. En la siguiente tabla se resumen algunas de las principales características de las dos alternativas que se tuvieron en cuenta para realizar el desarrollo de nuestra plataforma:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Licencias</td>
<td>Es gratis para desarrollo en PC.</td>
<td>Unity Free y Unity Pro</td>
</tr>
</tbody>
</table>
Tutoriales [http://unity3d.com/learn/tutorials/modules](http://unity3d.com/learn/tutorials/modules)  
| Compatibilidad Kinect | Si | Si, mediante un plug-in. |

El uso de XNA se debe a la determinación de usar herramientas de desarrollo compatibles con el entorno de Microsoft Windows y el SDK oficial de Kinect. Además de cumplir con el objetivo de representar objetos en 2D (representación en dos dimensiones) en los juegos que son parte de nuestra aplicación. Generalmente, los gráficos/objetos 2D se utilizan para pantallas de inicio, menús y HUD (Heads-Up Displays o cuadros emergentes), entre otras cosas.

**Box2d**

Es un motor de física de software libre escrito en C++, utilizado para la creación de videojuegos en 2D. El objetivo principal de dicha herramienta es manipular el movimiento de los objetos de manera realista y desarrollar juegos más interactivos.
Cocos2D

Es un motor de física de código abierto escrito en C++. Puede ser utilizado para construir juegos, aplicaciones y otros programas interactivos para diversas plataformas. El mismo optimizado para gráficos en 2D con OpenGL.

Chipmunk

Es un motor de física 2D simple, ligero, rápido y portátil escrito en C. Cientos de juegos han elegido este motor debido a la alta calidad, velocidad y precisión de sus simulaciones de física 2D.

En la siguiente tabla se resumen las principales características de las alternativas que se tuvieron en cuenta para realizar el desarrollo de nuestra plataforma:

<table>
<thead>
<tr>
<th></th>
<th>Soporte multilingüe</th>
<th>Licencia</th>
<th>Documentación</th>
<th>Compatibilidad XNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box2D</td>
<td>C++, Flash, Java, C#, JavaScript, Python.</td>
<td>zlib (open source)</td>
<td>Documentación técnica <a href="http://box2d.org/documentation.html">link</a> Foro desarrolladores <a href="http://box2d.org/forum/">link</a></td>
<td>Si</td>
</tr>
<tr>
<td>Cocos2D</td>
<td>C++, C#, JavaScript, Python, Objective-C, Swift.</td>
<td>Open source</td>
<td>Documentación técnica <a href="http://www.cocos2d-x.org/wiki">link</a> Foro desarrolladores <a href="http://discuss.cocos2d-x.org/">link</a></td>
<td>Si</td>
</tr>
<tr>
<td>Chipmunk</td>
<td>C++, JavaScript, Python, Objective-C, Ruby, Pascal.</td>
<td>MIT license (open source)</td>
<td>Documentación técnica <a href="http://chipmunk-physics.net/release/ChipmunkLatest-Docs/">link</a> Foro desarrolladores <a href="http://chipmunk-physics.net/forum">link</a></td>
<td>No</td>
</tr>
</tbody>
</table>

Después de realizar un análisis de las alternativas disponibles y teniendo en cuenta nuestra decisión de utilizar XNA para desarrollar la plataforma, la gran cantidad de ejemplos que existen en la web y la buena documentación, decidimos utilizar como motor de física Box2D.
El mundo bidimensional

La representación gráfica bidimensional usa dos de las tres dimensiones del espacio. Para representar los objetos empleamos el “ancho” y el “alto” y descartamos la profundidad. Dichas dimensiones se miden en píxeles en la pantalla en lugar de utilizar un sistema de coordenadas. De ésta manera, si deseamos dibujar un elemento en la mitad de la pantalla, tendremos que calcular, la posición basándonos en la pantalla del sistema, que puede o no tener diferentes configuraciones.

![Figura 8 - Coordenadas bidimensionales en píxeles](image)

Como se muestra en la imagen, la coordenada origen del sistema se encuentra en la esquina superior izquierda de la pantalla. El eje $x$ comienza en 0, y se desplaza hacia la derecha, mientras que el eje $y$ comienza en el mismo origen y se desplaza verticalmente hacia abajo.

Sprites

En el desarrollo de gráficos computacionales, un sprite es una imagen de dos dimensiones. La mayoría de los elementos gráficos que serán representados en dos dimensiones se hará mediante el uso de sprites, ya que la mayor parte del tiempo estaremos creando imágenes y objetos en pantalla en determinadas posiciones.
Por supuesto, la definición de los sprites no se limita sólo al dibujo de elementos estáticos. En el mundo bidimensional, a menudo se utilizan para lograr efectos especiales como explosiones, fuego, humo y objetos dinámicos en movimiento. Por ejemplo, podríamos utilizar una imagen (sprite) de un misil, en lugar de crear un modelo detallado en 2D.

En el siguiente pseudo-código se muestra cómo representar un sprite asociado a una imagen estática en pantalla:

```
1  Vector2 mPosition = new Vector2(X, Y);
2  Texture2D mSpriteTexture = <cargar textura>;
3  SpriteBatch sprite = new SpriteBatch(...);
4  sprite.Begin();
5  sprite.Draw(miTextura, miPosicion, Color.White);
6  sprite.End();
```

**Código 1 – Código renderización de un sprite estático.**

El resultado final de dicha implementación se puede observar en la siguiente figura:

![Figura 9 - Ilustración de un sprite estático en pantalla](image)

**Motor de física - conceptos centrales**

El objetivo principal de un motor de física es poder simular las mismas características físicas que en la vida real de los objetos creados o modelados, teniendo en cuenta por ejemplo gravedad, masa, fricción, restitución, etc.
Dicha característica permite a los videojuegos ser más atractivos al usuario final, como así también tener la capacidad de representar de manera aproximada efectos y situaciones que los usuarios realizan cotidianamente.

Por otro lado, para poder simular los objetos creados, se requiere un espacio bidimensional, donde contenga una colección de cuerpos y restricciones que interactúan entre ellos. Dicho espacio se denomina "Mundo físico", donde podemos destacar los siguientes conceptos:

**Shape**: Un shape es un objeto geométrico en 2D, el cual se utiliza para la detección de colisiones. Utilizamos dichos elementos para poder incluir la representación de formas como círculos o polígonos en un motor de física bidimensional.

**Rigid body**: Representa un trozo de materia rígida, donde la distancia entre dos rigid bodies es constante. Los Bodies son el eje fundamental de los fixtures, ya que pueden ser transportados por todo el mundo 2D. Por lo tanto, dos fixtures adjuntados al mismo body nunca se mueven uno respecto del otro.

**Fixture**: Representa un simple accesorio que permite relacionar un shape con un rigid body. También permite añadir propiedades del material tales como la densidad, la fricción, y la restitución. Un fixture relaciona un shape en el sistema de colisión de manera que pueda interactuar con otros shapes.

**Constraint**: Una constraint es una conexión física que elimina cierta libertad en los cuerpos. Por ejemplo, si tenemos un cuerpo 2D atado a un pin en la pared (como un péndulo), podemos limitar dicho cuerpo a la pared. En este punto el cuerpo sólo puede girar alrededor del pin, por lo que la restricción se ve reflejada en dicho pin.

**Contact constraint**: Es una restricción especial para prevenir la penetración entre objetos rígidos y simular la fricción y la restitución. En la mayoría de las implementaciones son creadas automáticamente por el motor de física.

**Joint**: Dicha restricción es usada para mantener dos o más rigid bodies juntos. Un motor de simulación de física soporta múltiples tipos de joints: revolute, prismatic, distance, entre otros.
Game Loop

La actividad principal de un juego se desarrolla dentro de un loop o bucle, que sólo deja de ejecutarse bajo determinadas condiciones ocasionadas por el usuario o generadas por el desarrollo dentro del juego.

Dentro de dicho bucle se realizan varias acciones en décimas de segundo que, en función del juego pueden efectuarse algunas o la mayoría de ellas, por ejemplo:

1. Comprobar eventos del usuario (se ha pulsado una tecla, se hizo clic, etc).
2. Mover los elementos del juego (el personaje principal, los enemigos, otros integrantes, ítems para recoger).
3. Detectar colisiones entre elementos (el personaje contra una pared que no puede atravesar, los proyectiles disparados contra los enemigos, un ítem que proporciona más vida, etc).
4. Dibujar el contenido completo de la pantalla en función de todo lo anterior (movimientos, saltos, disparos, muerte, heridas, fondos).

El siguiente pseudocódigo muestra una función Principal en la que dentro de un loop se determina constantemente su criterio de corte. En nuestro ejemplo, dicho criterio verifica si el juego en ejecución ha terminado o no.

Del mismo modo, dentro del loop se llama constantemente a diversas funciones que comprueban continuamente las condiciones del juego y actúan en consecuencia.

Se asume que todos los elementos necesarios durante el loop del juego se inicializan al comenzar la función Principal.
Código 2 – Pseudo código de un game loop.

El primero de los puntos dentro de un bucle de juego (línea 6), realiza constantemente una verificación de los controles que el usuario haya podido utilizar. Detecta pulsaciones del teclado o movimientos de mandos de juego y se debe vigilar todo aquello que sea susceptible de ser accionado por el jugador.

En el segundo punto (línea 7), en función del desarrollo del juego se ejecutan las acciones necesarias como pueden ser, mover el personaje hacia una dirección, saltar, disparar, realizar una pausa o abandonar la partida.

El tercero de los puntos (línea 8), actúa sobre las distintas colisiones que se puedan producir en el juego. Por ejemplo, un personaje manipulado por el jugador no debe poder atravesar una pared, por lo que el detector de colisiones debe determinar si se ha producido un toque entre ambos para anular los movimientos, dando la sensación de que se ha impactado con dicha pared. De la misma forma, el impacto de un proyectil sobre un enemigo es una colisión que debe ser detectada, como también el apoyo de los pies sobre el suelo o un golpe contra el techo al efectuar un salto.

El cuarto de los puntos del bucle (línea 9), se refiece a la representación del nuevo contenido atendiendo a todo lo anterior. La pantalla se dibuja varias veces por segundo, por lo que la sensación de movimiento es realista y fluida. En este punto se puede también poner en orden marcadores, número de vidas, monedas recogidas, tiempo restante de juego, etcétera.

Los bucles de juego pueden ser de paso fijo o variable. En el tipo de paso fijo, el juego intenta llamar a su método de actualización en un intervalo fijo y constante de tiempo. Por otro lado, los juegos de paso variable llaman a sus métodos de actualización en un bucle continuo.

En el entorno de desarrollo XNA [16] de Microsoft el tiempo de paso fijo que se emplea por defecto es del orden de 1/60.
Game Loop en XNA

Como ya mencionamos anteriormente, una de las principales herramientas utilizadas en el desarrollo de la plataforma es Microsoft XNA. Si bien XNA se implementa bajo el concepto de Game Loop, posee algunas características particulares como:

- El Game Loop no es tan visible como fue descrito anteriormente (ver sección Game Loop), sino que está implementado de manera tal que permanezca oculto para el programador. Del mismo modo, encontramos funciones que actualizan los estados de los elementos del juego, los renderizan en pantalla y se capturan acciones o eventos realizados mediante el hardware.
- Dentro de XNA se encuentra la clase Game que implementa un bucle de juego, el cual no solo proporciona la ventana que muestra la aplicación, sino que también proporciona métodos que pueden sobrecargarse y así facilitar la comunicación entre el juego y el sistema operativo.

A continuación, se muestra un diagrama de flujo que representa el proceso de ejecución de una aplicación XNA:
A continuación, se describen cada uno de sus métodos:

- **Initialize**
  
  En dicho método se inicializan todas las variables y estructuras necesarias para iniciar la aplicación. Del mismo modo, se configuran las propiedades de todos los
dispositivos utilizados de manera de asegurar su adecuado funcionamiento (dispositivo Kinect en nuestro caso).

- **Load Content y Unload Content**
  Son utilizados para cargar y liberar recursos multimedia que serán empleados dentro de la aplicación, como por ejemplo imágenes, sonidos, texturas, etc. Debido a la naturaleza de los recursos, es conveniente cargarlos al inicio de la aplicación, para luego liberar memoria y espacio utilizado en el sistema operativo.

- **Update**
  En el método Update es dónde se actualizan las variables, se lleva el control de los eventos, se controlan las colisiones, las animaciones, los cambios de posición, etc. Prácticamente es dónde se lleva el seguimiento y el control de todo el juego.
  Para entender este método es necesario comentar que recibe el parámetro `GameTime`. Literalmente significa tiempo de juego, su valor es variable según el procesador e indica la velocidad de refresco del juego en la aplicación de XNA. Es decir, cada cierto instante de tiempo, se actualiza todo su contenido.

- **Draw**
  Al igual que `Update`, recibe el parámetro `GameTime`. En este método se posicionan los objetos, texturas e imágenes que se irán dibujando en cada tiempo de juego y se actualiza a la misma velocidad que el método Update (cada `GameTime`).

Para demostrar la estructura básica con la que se cuenta dentro de XNA, a continuación, se explicará un breve ejemplo de la estructura de un videojuego muy simple hecho en XNA C#:

```csharp
1 public class GameEjemplo : Microsoft.Xna.Framework.Game
2 {
3     GraphicsDeviceManager graphics;
4     SpriteBatch spriteBatch;
5     // Esta es la textura a dibujar.
6     Texture2D myTexture;
7     // Se definen las coordenadas del sprite en (0,0).
8     Vector2 spritePosition = Vector2.Zero;
9     ...
10 }
```

*Código 3 – Ejemplo declaraciones en XNA.*

En el código anterior podemos observar la declaración de los elementos `GraphicsDeviceManager` y `SpriteBatch` (líneas 3 y 4 respectivamente), los mismos son utilizados para realizar la renderización en pantalla y ya son elementos predefinidos por XNA. También encontramos otros elementos declarados que son parte del propio juego,
tales como `myTexture` y `spritePosition` (líneas 6 y 8), que determinan el objeto textura y su posición respectivamente.

```csharp
11 ...
12 public GameEjemplo()
13 {
14     graphics = new GraphicsDeviceManager(this);
15     Content.RootDirectory = "Content";
16 }
17 ...
```

**Código 4 – Ejemplo de inicializaciones en XNA.**

Dentro del constructor de `GameEjemplo` (línea 14) se puede ver la inicialización del elemento `GraphicsDeviceManager` y también se define cuál será el directorio del sistema en donde está almacenado todo el contenido necesario, tales como fondos, sonidos y sprites.

```csharp
18 ...
19 protected override void Initialize()
20 {
21     // Agregar las inicializaciones de elementos y variables lógicas aquí.
22     base.Initialize();
23 }
24
25 protected override void LoadContent()
26 {
27     // Crea un nuevo SpriteBatch, el cual se usa para dibujar texturas.
28     spriteBatch = new SpriteBatch(GraphicsDevice);
29     myTexture = Content.Load<Texture2D>("mytexture");
30 }
31
32 protected override void UnloadContent()
33 {
34     // Elimina el contenido de Content
35 }
36 ...
```

**Código 5 – Ejemplo métodos básicos de XNA.**

Además, para ejemplificar el flujo de ejecución mostrado en la **Figura 5**, podemos observar el método `Initialize()` (línea 19) en donde se inicializan elementos de la lógica del juego, luego se carga contenido necesario en memoria en el método `LoadContent()` (línea 25) y
finalmente se elimina de la memoria el contenido multimedia cuando el juego termina en el método `UnloadContent()` (línea 32).

```
...  
protected override void Update(GameTime gameTime)
{
    // Si el jugador presiona la tecla escape se termina la ejecución.
    if (Keyboard.GetState().IsKeyDown(Keys.Escape)) {
        this.Exit();
    }
    // Mueve el sprite por la pantalla.
    updateSprite(gameTime);
    base.Update(gameTime);
}
protected override void Draw(GameTime gameTime)
{
    graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
    // Se dibuja el sprite.
    spriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend);
    spriteBatch.Draw(myTexture, spritePosition, Color.White);
    spriteBatch.End();
    base.Draw(gameTime);
}
```

**Código 6 – Ejemplo métodos básicos de XNA.**

Por otra parte, los módulos `Update` y `Draw` representados en el flujo de ejecución son implementados en los métodos `Update()` y `Draw()` (líneas 38 y 50). El primero de ellos es el encargado de realizar la actualización de posiciones del sprite y capturar eventos del teclado. Mientras que en el método `Draw()`, mediante el uso del elemento `SpriteBatch` de XNA, podemos graficar nuestra textura definida anteriormente, en la posición de pantalla en que se encuentra el sprite (líneas 6. y 8. respectivamente).

Cabe destacar el uso del objeto `GameTime` ya que es utilizado por los métodos `Update()` y `Draw()`. Dicho elemento representa el tiempo transcurrido desde la última llamada para actualizar o dibujar, dependiendo del caso.
Diseño de la aplicación

Dentro del marco de tecnologías ya definidas y de los conceptos generales previamente ya explicados, comenzamos a describir cómo es el diseño de la plataforma, la cual permite desarrollar videojuegos incorporando el uso de Kinect, un motor de física y gráficos en dos dimensiones.

En busca de entender el diseño de manera simple, se irá describiendo paso a paso cómo fue presentado el desarrollo de la misma. Al principio comenzaremos describiendo cómo se grafican los elementos del juego, cómo se incorpora la física a cada uno de ellos y finalmente, cómo se representa el cuerpo del usuario con los datos obtenidos desde el dispositivo Kinect.

Representación gráfica de elementos

Como se mencionó anteriormente, los juegos se desarrollarán en un espacio de dos dimensiones. Para comenzar a desarrollar un videojuego simple, primero se necesita un espacio donde contener todos los elementos involucrados en una simple escena, pantalla o menú. De esta manera, surge en el diseño de la plataforma la clase **GameScreen**, que representa una simple pantalla o capa que contiene lógica suficiente de actualización y renderización.

Una de las principales responsabilidades es ser la encargada de contener los elementos de un videojuego.

Adicionalmente, **GameScreen** posee las siguientes responsabilidades fundamentales para el funcionamiento de un videojuego:

- Crear e inicializar todas las variables y estructuras involucradas.
- Cargar todos los elementos gráficos necesarios, tales como imágenes de fondo y texturas.
- Permitir el ingreso de datos de entrada por parte del usuario, sólo en ocasiones cuando la pantalla está activa.
- Cada cierto tiempo se debe actualizar las propiedades de los elementos dentro del videojuego.
- Cada cierto tiempo se debe indicar a los elementos que intervienen en la pantalla que deben dibujarse.

Dichas responsabilidades se explicarán en detalle, pero en principio nos centraremos en la representación de los elementos en pantalla. A continuación, se muestra la representación de la clase **GameScreen**: 

---

37
Ya definida la clase que contiene todos los elementos de un videojuego, es necesario poder representar cada uno de ellos. De esta forma surge en el diseño la clase **DrawableElement**. La misma es la encargada de dibujar el elemento en una posición determinada de la pantalla.

Para hacer más gráfica la explicación, vamos a suponer que se desea crear un elemento circular dentro de una pantalla, que por ahora no realizará nada más que dibujarse dentro de la misma. De ésta manera, dentro de la clase **GameScreen** encontraremos el código que inicializa el elemento y lo dibuja:

```java
1   public class GameScreen {
2       ...
3       public void initialize() {
4           DrawableElement solidCircle = new ElementCircle(radius, color, alpha);
5       ...
6   }
7   
8   public void draw(GameTime gameTime) {
9       ...
10      solidCircle.draw(spriteBatch, basicEffect);
11   }
12 }
```

**Código 7 – Ejemplo renderización de elemento circular.**
Dando como resultado la siguiente ilustración:

![Ilustración de un círculo en pantalla](image)

**Figura 12** – Representación de un círculo en pantalla

En este ejemplo se representa el círculo con la clase **DrawableElement**, la cual estará contenida por la clase **GameScreen**, la misma es la encargada de indicarle a la clase **DrawableElement** que debe dibujarse mediante el método **draw**, el cual se encarga de dibujar un círculo mediante el uso de primitivas 2D.

![Diagrama de clases incluyendo a DrawableElement](image)

**Figura 13** – Diagrama de clases incluyendo a **DrawableElement**

Suponiendo que ahora se necesita sumar un elemento más a **GameScreen**. Se incorporará una caja que será representada en pantalla con una textura. Esta forma de representar elementos aún no está contemplada en el diseño. Por esta razón, se debe abstraer comportamiento de la clase **DrawableElement** y surgen, heredando comportamiento de ésta, las clases **ElementCircle** y **ElementTexture**. Las mismas representan en pantalla un círculo y una textura dentro del mundo del videojuego.
Como ya se mencionó previamente, las clases ElementCircle y ElementTexture heredan el método draw, con el cual son dibujados en pantalla cuando la clase GameScreen les indica a ambos que se dibujen.

En el siguiente código de la clase GameScreen, en el método initialize y draw se encuentra la inicialización y renderización del elemento círculo y caja:

```java
public class GameScreen {
    private DrawableElement solidCircle, textureBox, line;

    public void initialize() {
        solidCircle = new ElementCircle(radius, color, alpha);
        textureBox = new TextureElement(...);
    }

    public void draw(GameTime gameTime) {
        solidCircle.draw(spriteBatch, basicEffect);
        textureBox.draw(spriteBatch, basicEffect);
    }
}
```

Código 8 – Ejemplo renderización de un círculo y un sprite.
Dicho ejemplo se representa de la siguiente manera:

![Representación de un círculo y una caja en pantalla](image)

**Figura 15** – Representación de un círculo y una caja en pantalla

Siguiendo con la representación de elementos en pantalla, además del círculo y la caja, se incorpora al videojuego una línea. Dicha representación servirá más adelante para dibujar al jugador dentro del videojuego.

![Diagrama de clases incluyendo maneras adicionales de dibujar una figura. En este caso se incluyen ElementCircle y ElementPolygon](image)

**Figura 16** – Diagrama de clases incluyendo maneras adicionales de dibujar una figura. En este caso se incluyen ElementCircle y ElementPolygon
Con esta última incorporación se abstraen todas las características comunes entre los elementos, creando la clase `ElementShape`, que permitirá representar en pantalla elementos tales como un círculo, una línea o polígonos.

De esta manera, cada elemento hereda la forma de representar mediante el método `draw`. Dicha implementación dependerá de las características propias de cada objeto, dependiendo de la forma y del tipo de representación, ya sea utilizando primitivas o texturas.

En el siguiente código se muestra la inicialización de cada uno de los elementos representados en la imagen anterior y además se puede ver como es el funcionamiento del método `draw` que grafica cada uno de los elementos ya mencionados.

```java
public class GameScreen {
    private DrawableElement solidCircle, textureBox, line;

    public void initialize() {
        solidCircle = new ElementCircle(radius, color, alpha);
        textureBox = new TextureElement(...);
        line = new ElementPolygon(x, y, //
                        width, height, backGroundColor, alpha, isSolid);
    }

    public void draw(GameTime gameTime) {
        solidCircle.draw(spriteBatch, basicEffect);
        textureBox.draw(spriteBatch, basicEffect);
        line.draw(spriteBatch, basicEffect);
    }
}
```

*Código 9 – Ejemplo renderización de todos los elementos.*
Dicha implementación se representa de la siguiente manera:

![Figura 16 – Representación de un círculo, una caja y una línea en pantalla](image)

Representación física de la interacción entre elementos

Considerando los conceptos vistos en la sección **Motor de física - Conceptos generales**, explicaremos las variaciones que surgieron en el diseño en la incorporación de física a los elementos que intervienen en el videojuego.

Siguiendo los ejemplos anteriores, a los elementos ya previamente dibujados, se le va a sumar movimiento y distintos comportamientos. Para esto, debemos incorporar un “mundo físico” en donde los elementos interactúan entre sí, que se crea mediante la utilización de **Box2d**. De ésta manera vemos la incorporación del atributo **physicWorld** en la clase **GameScreen**.

![Figura 17 – Diagrama de clase de GameScreen](image)

**GameScreen** como se había visto antes, debe hacer que todos los elementos que contiene se dibujen cuando sea necesario. Adicionalmente, se le suman la responsabilidad de
actualizar y chequear los estados de cada uno de los elementos. Por esto mismo, encontramos en la clase `GameScreen` el método `update`.

Como se puede ver en el siguiente fragmento de código, en el método `update` se llama a un método que realiza verificaciones sobre el estado del juego, controlando si se terminó el tiempo de juego o si cumplió los objetivos del mismo.

```csharp
public class GameScreen {
    private List<GameElements> elements;
    private currentTimeSpan;
...
    public void update(GameTime gameTime) {
        verifyGameStatus();
        currentTimeSpan = TimeSpan.FromMilliseconds(5000);
        ...
    }

    public void verifyGameStatus() {
        int state = getPlayerState();
        if (state == 1) {
            // Si gana el juego.
        } else if (state == -1) {
            // Si pierde el juego.
        }
    }

    public override int getPlayerState() {
        // Se asume con state = 0 que el jugador siempre sigue jugando.
        int state = 0;
        int kicks = getCountKicks();

        if (currentTimeSpan.Minutes == TimeSpan.Zero.Minutes)
        {
            // Pierde si se acaba el tiempo.
            state = -1;
        } else if (kicks == elements) {
            // Gana si golpea todos los elementos.
            state = 1;
        }
        return state;
    }
}
```

*Código 10 – Validación del estado del juego.*
Enfocando nuestra atención a la interacción de los elementos, se debe encontrar una forma para representarlos dentro de GameScreen. Para éste fin se propone como solución en el diseño la clase GameElement, la cual representa los elementos dentro del mundo del videojuego. Esto implica que dentro de ella se maneja el comportamiento del elemento dentro del videojuego, posición en donde se encuentra, color, entre otras propiedades posibles.

Para fusionar la representación y la lógica de cada elemento, dentro de cada GameElement en el método display se utiliza una clase ya definida previamente, DrawableElement. Con esto último se logra poder representarse a sí mismo cuando el GameScreen lo requiere.
Resumiendo un poco la interacción entre estas dos clases, GameScreen tiene un método draw en donde se le indica que se dibujen a cada uno de los elementos que contiene, mediante el método display. Además, tenemos un método update en donde se actualizan las posiciones de cada uno de los elementos y el estado de cada uno de ellos, ya que un elemento de un instante para otro puede desaparecer del videojuego o cambiar su color.

Figura 18 – Diagrama de clases que representa la relación de dependencia entre GameScreen y GameElement
Figura 19 – Diagrama de clases que representa la jerarquía de elementos cuyo comportamiento es dibujarse en pantalla

Todos los últimos cambios que se fueron incorporando en el diseño permiten que los elementos dentro del mundo del videojuego interactúen entre sí. Luego se verán gráficamente en pantalla los cambios de la interacción que resulte entre ellos.

En el siguiente fragmento de código se puede ver cómo interviene la clase GameElement que se incorporó al diseño y cómo se la utiliza desde la clase GameScreen.
Representación de elementos específicos del juego

Como se comentó en secciones anteriores, un elemento tiene la capacidad de representarse de forma gráfica y también puede interactuar de forma física dentro del mundo del videojuego. En esta sección se explican los cambios que se introducen en el diseño de la plataforma; los cuales permiten definir las respectivas lógicas de los elementos que le dan a cada uno de los elementos sus propios comportamientos dentro del videojuego.

Ahora se debe incorporar los diferentes elementos de un videojuego con sus respectivas lógicas. Si buscamos representar distintos elementos dentro del videojuego, debemos definir cada uno de sus comportamientos dentro de él. Continuando con el ejemplo en el que se grafica en pantalla con primitivas un círculo y una caja mediante una textura 2D.

Ahora le asignaremos a cada uno de esos elementos un rol que cumplirán dentro del juego. Esto quiere decir que vamos a encontrar una gran diferencia en los comportamientos dentro del juego de los dos GameElement que forman parte del GameScreen. Cada uno de los GameElement va a tener su forma de representarse en pantalla, su física de movimiento y su comportamiento dentro del juego. Por esta razón podemos diferenciar las clases Circle y la clase Box que heredan comportamiento de la clase GameElement. Entre estas dos
clases se puede encontrar un método que tienen en común, display. Este método si bien lo encontramos en ambas clases, se implementan en cada una de ellas de diferente forma, ya que cada cual tiene formas diferentes de graficarse en pantalla. También comparten atributos como por ejemplo la posición que ocupan en pantalla o el color que poseen.

Figura 20 – Diagrama de clases que representa la jerarquía de GameElement.
En el siguiente fragmento de código, se puede ver cómo se utilizan las nuevas clases que fueron incorporadas al diseño.

```java
public class GameScreen {
    GameElement circle, box;
    ...
    public void initialize() {
        circle = new Circle(position, radius);
        box = new Box(point1, point2, point3, point4);
    }
    ...
    public void update(GameTime gameTime) {
        verifyGameStatus();
        ...
    }
    public void draw(GameTime gameTime) {
        circle.display(spriteBatch, basicEffect);
        box.display(spriteBatch, basicEffect);
    }
}
```

Código 12 – Incorporación y representación del elemento Box.

Representación de jugadores

Dentro del diseño que se describió hasta el momento se tiene la posibilidad de incorporar diferentes elementos al juego que pueden interactuar entre ellos, pero aún no hicimos foco en cómo se representa un jugador dentro del videojuego. Primero se deben repasar algunos conceptos principales sobre Kinect para luego poder centrar la atención en cómo se representa el jugador. El hardware de Kinect y el SDK reconocer el cuerpo de la persona que está frente, luego de hacer el procesamiento correspondiente, se obtienen como resultado un conjunto de 20 puntos cuando el jugador está de pie y 10 puntos en el caso que se utilice el modo de jugador sentado.
Estos puntos representan una parte del cuerpo específica, por ejemplo, podemos encontrar entre ellos el punto que representa la cabeza del jugador, el cuello, el hombro, cada una de las rodillas, etc. Cada uno de estos puntos va a tener una posición tridimensional \((x, y, z)\) pero en nuestro caso, al ser juegos de dos dimensiones tendremos mapeada en pantalla coordenadas \((x, y)\). En cada movimiento del jugador, las posiciones \((x, y)\) de los puntos mapeados cambian, por lo tanto, obtenemos un nuevo conjunto de puntos con las nuevas posiciones.
Una vez que se obtiene el conjunto de puntos llegamos al momento de representarlos gráficamente y físicamente dentro del mundo del videojuego. A partir de esta necesidad surge en el diseño la clase Skeleton, hereda comportamiento de la clase GameElement. Podemos encontrar entre sus principales responsabilidades:

- Almacenar el conjunto de puntos correspondiente al cuerpo del jugador.
- Contiene un conjunto de elementos del tipo Circle, que luego van a servir para representar de forma gráfica en pantalla y de forma física dentro del mundo creado por el motor de física.
- Actualizar cada uno de los puntos del cuerpo del jugador.
- Devolver la posición (x, y) de alguno de los puntos que forman parte del conjunto.
- Realizar la representación del cuerpo del jugador en pantalla, dibujando cada uno de los círculos que representan los puntos del cuerpo del jugador.

**Figura 23 – Diagrama de clases de jerarquía de GameElement.**
En el siguiente fragmento se muestra el funcionamiento de la clase GameScreen utilizando a Skeleton para representar el cuerpo del jugador, como se muestra en la imagen previa:

```java
public class GameScreen {
    GameElement circle, box;

    public void initialize() {
        circle = new Circle(position, radius);
        box = new Box(point1, point2, point3, point4);
    }

    public void update(GameTime gameTime) {
        verifyGameStatus();
    }

    public void handleInput(GameTime gameTime, InputState input) {
        ...
        skeleton.update(KinectSDK.Instance.getJoints());
    }

    public void draw(GameTime gameTime) {
        ...
        skeleton.display(spriteBatch, basicEffect);
    }
}
```

Código 13 – Representación del esqueleto del usuario mediante el objeto Skeleton.

Interacción con Kinect

El SDK de Kinect es frecuentemente actualizado agregando nuevas funcionalidades y mejorando otras que ya existen. Esto da como resultado que algunas veces las llamadas a ciertos métodos en versiones consecutivas cambien los nombres de los métodos o los parámetros que reciben. Esta situación provoca la necesidad de aislar las llamadas nativas del SDK en una clase en donde no importe si cambia o no la implementación de un método, dentro del código del juego siempre se va a invocar al método requerido de la misma forma. Con la clase KinectSDK se busca que todas las llamadas necesarias al SDK no estén dispersas por todo nuestro código y no sea necesario realizar cambios cuando los nombres de los métodos sean cambiados o cuando a los tipos o nombres de parámetros les ocurra algo similar. KinectSDK además es la encargada de guardar una instancia propia de sí misma, con esto se busca que en toda la aplicación se utilice una única instancia de la clase y evitar múltiples instancias de la misma, el acceso a la instancia de la clase se hace invocando el método getInstance. Este tipo de comportamiento se obtiene cuando se aplica un patrón de diseño llamado Singleton [17].
Dentro de la clase podemos encontrar varios atributos:

- La resolución de pantalla en que se está trabajando para ajustar el esqueleto del jugador.
- La instancia de la propia clase.
- El estado de conexión del dispositivo.
- El conjunto de puntos que hacen referencia al cuerpo del jugador.
- Una textura 2D que contiene las muestras que se van tomando de la cámara RGB.
- La resolución de la pantalla.

Entre los métodos disponibles podemos encontrar:

- Inicializar el dispositivo.
- Devolver la instancia con que se está utilizando.
- Cambiar el estado del dispositivo.
- Detectar los frames que se proveen las cámaras de profundidad de puntos y RGB, que detectan los puntos del esqueleto.
- Escalar el esqueleto a la resolución que se necesita.
- Graficar en pantalla los datos que se obtienen de la cámara RGB de Kinect.
- Realizar la graficación de las articulaciones y los huesos del jugador.
Figura 24 – Diagrama de clases que muestra la relación de KinectSDK, GameElement y GameScreen.

En el siguiente fragmento de código se muestra como se realiza la actualización y visualización de los puntos que representan al jugador, utilizando la clase KinectSDK dentro de las clases GameScreen y Skeleton.
public class GameScreen {
    ...

    public void handleInput(GameTime gameTime, InputState input) {
        ...
        skeleton.update(KinectSDK.Instance.getJoints());
    }

    public void draw(GameTime gameTime) {
        ...
        skeleton.display(spriteBatch, basicEffect);
    }
}

public class Skeleton {
    ...

    public void update(IDictionary<Microsoft.Kinect.JointType, Vector2> js) {
        ICollection<Microsoft.Kinect.JointType> keys;
        if (js != null && circleJoints.Count > 0) {
            keys = js.Keys;
            for (int i = 0; i < CommonConstants.JOINTS_COUNT; i++) {
                circleJoints[i].getBody().Position = //
                js[keys.ElementAt<Microsoft.Kinect.JointType>(i)] /
                CommonConstants.PIXELS_TO_METERS;
            }
            jointTypes = js;
        }

        public void display(SpriteBatch spriteBatch, BasicEffect basicEffect) {
            ...
            KinectSDK.Instance.display(spriteBatch, basicEffect);
        }
    }
}

Código 14 – Actualización constante de los joints del esqueletos y representación de los mismos.

Manejo de escenas

Simon proporciona una base de distintas pantallas para que el desarrollo de un videojuego sea más rápido. Hasta el momento nos centramos únicamente en la pantalla en donde se desarrolla la acción del juego. A partir de ahora se mostrarán cada una de las pantallas que existen en el diseño de la plataforma.

Como se vió en la secciónRepresentación gráfica de elementos, la clase GameScreen contenía al juego en sí mismo, en esta sección se explican los cambios que se introdujeron para permitir tener otros tipos de pantallas tales como un menú de opciones, un menú de pausa, etc.
La clase **GameScreen** sigue teniendo métodos para:

- Inicializar las estructuras de los elementos necesarios.
- Cargar y descargar los elementos gráficos.
- Actualizar los elementos de pantalla.
- Dibujar cada uno de los elementos en pantalla.
- Capturar interacción mediante periféricos de entrada.

Las clases que extienden de **GameScreen** heredan los siguientes métodos, los cuales serán redefinidos en caso de ser necesario.

![Diagrama de clases de GameScreen](image)

**Figura 25 – Diagrama de clases de GameScreen.**

A partir de ahora se pueden diferenciar por lo menos dos tipos de pantallas. Una que contiene un menú principal y otra pantalla que contiene el juego en sí mismo.

A modo de ejemplo gráfico veremos una pantalla de menú principal simple con algunas opciones disponibles.
La clase **MenuScreen** hereda comportamiento de **GameScreen** y posee otros atributos y métodos adicionales necesarios para mostrar y seleccionar opciones del menú, diferenciándose de otro tipo de pantalla. Esta clase tiene como diferencias principales a las demás:

- Una lista de los elementos que forman parte del menú.
- Cuando se dibuje en pantalla, se deben mostrar los elementos que son parte del menú, la información del jugador y la información del juego seleccionado.
- Cuando se realiza la actualización, se deben actualizar los estados de los elementos que son parte del menú.
- Se deben capturar los eventos de teclado que sirven para seleccionar las opciones del menú.
Otro tipo de pantalla que se puede diferenciar dentro del diseño de la plataforma es aquella que contiene el juego por sí mismo, la clase `GamePlayScreen` tiene como responsabilidades:

- Inicializar todas las variables del videojuego.
- Actualizar las variables necesarias para el funcionamiento.
- Realizar los distintos chequeos para determinar si se gana, se pierde o si se debe seguir jugando, los cuales dependen de la misma lógica del juego.
- Representar los elementos que intervienen en el videojuego.

Entre los atributos más importantes dentro de ésta clase se puede encontrar:

- El mundo físico en donde se desarrolla el videojuego.
- La lista de elementos que intervienen.
- Los elementos que representan al jugador (esqueleto).

En el Capítulo 4: Instanciación, se describe en detalle el comportamiento y la utilidad de la clase `GamePlayScreen`.

**Figura 27 – Diagrama de clases que muestra la jerarquía de GameScreen.**
Así como surgieron las clases que hacen posible tener un menú principal u otra clase que se dedica íntegramente al videojuego en sí mismo; surge también la necesidad de incorporar un menú de pausa. Por esta razón se decidió que la clase MenuScreen sea una clase abstracta que provee métodos comunes entre el menú de opciones que se ofrece en el menú principal y el menú de pausa que se provee cuando el jugador interrumpe el videojuego. Para incorporar al diseño estos dos nuevos tipos de pantallas, se extendieron dos clases: PauseMenuScreen y MainMenuScreen.

La clase MainMenuScreen hereda los principales métodos de la clase MenuScreen y se le suman los métodos específicos del menú principal. Entre sus principales responsabilidades se pueden mencionar:

- Permite iterar entre los diferentes jugadores para seleccionar el perfil del jugador.
- Permite iterar entre los diferentes juegos incorporados en la aplicación.
- Maneja los eventos de teclado para cada una de las opciones del menú.
La clase `PauseMenuScreen` también hereda los principales métodos de la clase `MenuScreen` y se le suman otros métodos específicos para el comportamiento que debe tener el menú de pausa de un videojuego. Entre ellos podemos encontrar:

- Manejo de eventos para cada una de las opciones del menú de pausa.
- Confirmar la selección del usuario de salir del videojuego.

**Administrador de pantallas**

En esta sección se describe la incorporación de la clase `ScreenManager` en el diseño de la plataforma, dicha clase tiene como principal responsabilidad realizar las transiciones de una pantalla a otra. Dentro de las funcionalidades que se pueden encontrar en ésta clase, se destacan:

- Agregar una pantalla a la lista.
- Eliminar una pantalla de la lista.
- Actualizar la pantalla actual.
- Representar los componentes de cada pantalla.
- Cargar y descargar el contenido necesario para realizar la representación en pantalla.
- Retornar el alto y el ancho de pantalla que se utiliza en el momento para realizar los cálculos previos a la renderización del contenido de la pantalla a representar.
Dicha clase tiene una lista de todas las pantallas disponibles para hacer la transición, por ejemplo: en el momento en el que un juego es pausado, el `ScreenManager` hace la transición hacia la pantalla que muestra el menú de pausa con las opciones disponibles.

**Figura 30 – Diagrama de clases que muestra la relación entre GameScreen y ScreenManager**

En el siguiente código se puede observar la inicialización de la clase `ScreenManager` y luego se agrega la pantalla `MainMenuScreen` a las pantallas que dispone el mismo, que permite mostrar una pantalla de menú principal cuando se inicia el juego.

```csharp
public class Game : Microsoft.Xna.Framework.Game
{
    public Game()
    {
        screenManager = new ScreenManager(this);
        Components.Add(screenManager);
        AddInitialScreens();
    }

    private void AddInitialScreens()
    {
        screenManager.AddScreen(new MainMenuScreen(), null);
    }
}
```

**Código 15 – Incorporación de una nueva pantalla al objeto ScreenManager.**
Capítulo 4

Instanciación

Cada videojuego posee un objetivo en particular que depende de las necesidades planteadas por los docentes del instituto. Puede resultar tanto o más enriquecedor que cualquier otra actividad siempre y cuando el mismo ofrezca oportunidades de aprendizaje y sea abordado con la orientación necesaria para aprovechar su valor educativo.

Los videojuegos resultan potentes estrategias metodológicas que favorecen el aprendizaje significativo y constructivo en un entorno lúdico y digital que tanto motiva e interesa a los alumnos.

Como se explicó en el capítulo anterior, Simon es una plataforma de desarrollo de videojuegos que permite la implementación de aplicaciones asociadas a diferentes tecnologías. Los videojuegos desarrollados dentro de la plataforma son el resultado del trabajo en conjunto con el instituto ATAD (Asociación Tandilense de Ayuda al Discapacitado) de la ciudad de Tandil, donde se realizó la toma de requerimientos de los diferentes juegos. Una vez desarrollados, los videojuegos fueron evaluados en el mismo instituto con la participación de docentes y alumnos.

En ATAD se buscó ofrecer experiencias potencialmente educativas al permitir al alumno:

- Adquirir nuevos conocimientos.
- Desarrollar habilidades psicomotrices, así como la coordinación mano-vista.
- Desarrollar habilidades de pensamiento crítico, estrategia y toma de decisiones.
- Desarrollar actitudes de superación y autoestima.
- Aprender a compartir y colaborar con el otro.
- Potenciar la fantasía, la imaginación y la creatividad.

De la implementación del framework se destaca la facilidad de uso de la plataforma y la adaptabilidad de la misma. Es decir, que permite crear diversos juegos en base a propiedades particulares. Por lo tanto, el desarrollador sólo debe ocuparse de la lógica de un juego determinado más que de la anatomía del framework. Dicha ventaja permite desarrollar juegos con un alto valor educativo y lúdico.

En éste capítulo se hace una descripción detallada de cada uno de los videojuegos que fueron desarrollados dentro de la plataforma, se muestra paso a paso la facilidad con que
se desarrollaron los videojuegos que fueron instanciados y cómo pueden adaptarse a las necesidades que los docentes y los niños requieren.

**Videojuego Círculos**

En este videojuego el jugador debe ir golpeando los círculos que se desplazan y rebotan por toda la pantalla. Con un primer contacto el círculo cambia de color y al segundo contacto desaparece de la pantalla. La cantidad inicial de círculos y la velocidad en que se mueven los mismos depende del nivel de avance en el que se encuentre el jugador.

![Figura 31 – Captura del juego completo Círculos.](image)

Para comenzar a desarrollar el videojuego **Círculos**, consideremos el siguiente diagrama de clases:
Figura 32 - Diagrama de clases representando el objeto encapsulador `GamePlayScreen` para videojuegos.

Como se muestra en el diagrama anterior, se debe extender de la clase abstracta `GamePlayScreen`, que nos permite heredar el siguiente comportamiento:

- Creación de elementos.
- Actualización de elementos y chequeos de estados.
- Renderización de elementos en pantalla.
Incorporación del jugador al videojuego

El primer paso es incluir la funcionalidad de detección de esqueleto del usuario. Esto se realiza invocando el método `createSkeletonWithPhysic` de la clase `GamePlayScreen`, dentro de la clase `GamePlayScreenCircles`:
class GameplayScreenCircles : GameplayScreen
{
    public GameplayScreenCircles(...) {
        // inicialización e incorporación a la simulación física.
        createSkeletonWithPhysic(physicsWorld);
        ...
    }
    ...
}

Código 16 – Creación del esqueleto del jugador e incorporación al mundo físico.

La inicialización de un objeto Skeleton permite relacionar los elementos que conforman el esqueleto con un mundo físico, el cual se describe en detalle en la sección Incorporación de simulación física.

El segundo paso es mantener actualizado las coordenadas que conforman el esqueleto del usuario. Dicha detección se realiza dentro del método HandleInput que heredamos de la clase GameplayScreen. HandleInput nos permite manipular la entrada de información por parte del usuario, que a diferencia del método Update, sólo es invocado cuando la pantalla está activa (otras pantallas no poseen el foco):

class GameplayScreenCircles : GameplayScreen
{
    ...
    public GameplayScreenCircles(...) {
        // inicialización e incorporación a la simulación física.
        createSkeletonWithPhysic(physicsWorld);
        ...
    }
    ...

    public override void HandleInput(...) {
        ...
        updateSkeletonJoints();
        ...
    }
    ...
}

Código 17 – Invocación a la actualización de los puntos que conforman el esqueleto del jugador.
Código 18 – Actualización de los puntos que conforman el esqueleto, delegando dicho comportamiento al objeto `Skeleton`.

Como se puede observar en la línea 5, la actualización de los puntos del esqueleto debe obtenerse mediante el uso del método `getJoints` de la clase utilitaria `KinectSDK`.
Por último, para visualizar el esqueleto, debemos representar todos los puntos pertenecientes al mismo. Para ello dentro del método heredado Draw llamamos a representar el esqueleto del usuario de la siguiente manera:

```csharp
1  class GameplayScreenCircles : GameplayScreen
2  {
3      ...
4          public GameplayScreenCircles(...) {
5              // inicialización e incorporación a la simulación física.
6              createSkeletonWithPhysic(physicsWorld);
7          }
8
9          public override void HandleInput(...) {
10              ...
11              updateSkeletonJoints();
12          }
13
14          public override void Draw(...) {
15              ...
16              // Muestra el cuerpo del jugador si está dentro del rango de juego.
17              if (isSkeletonInRange()) {
18                  displaySkeleton();
19              }
20          }
21      }
22  }
```

_Código 19 – Representación de los puntos que conforman el esqueleto._

Como se puede observar en el desarrollo parcial del método Draw, el esqueleto del usuario se representará siempre y cuando se respete el rango definido para su representación. Nuevamente, dicha validación se realiza mediante el método _isInRange_ de la clase utilitaria KinectSDK. Por otra parte, la representación del esqueleto del usuario se delega al objeto Skeleton:
class GameplayScreen : GameScreen {
  ...
  private Skeleton skeleton;

  public boolean isSkeletonInRange() {
    return KinectSDK.Instance.isInRange();
  }

  public void displaySkeleton() {
    ...
    skeleton.display(...);
  }

  ...
}

Código 20 – Representación de los puntos que conforman el esqueleto, delegando dicha funcionalidad al objeto Skeleton.

Los pasos anteriores pueden verse resumidos en la siguiente figura:

Figura 34 – Representación del esqueleto.

Incorporación de simulación física

Adicionalmente, se debe incluir la funcionalidad para la simulación física del videojuego (que hará que reboten las pelotas en pantalla, por ejemplo). Para ello, debemos invocar el método createPhysicWorld de la clase GameplayScreen:
Una vez inicializado el mundo físico, debemos iniciar la simulación mediante llamadas repetidas al método `Step` (perteneciente al objeto `World` de `Box2D`). Dicha invocación se realiza desde el método `Update`:

```csharp
public override void Update(...) {
    if (isSimulationActive()) {
        simulateWorld();
    }
    ...
}
```
Como puede observarse la simulación física del mundo se llevará a cabo dependiendo del estado de simulación actual de la aplicación. El método `isSimulationActive` retorna dos estados posibles, estará en `true` cuando el mundo físico está en plena simulación y los elementos se encuentran en movimiento interactuando entre sí. Por otra parte, tendrá valor `false`, cuando la simulación está desactivada y los elementos del mundo no poseen movimiento alguno:

```java
1  class GameplayScreen : GameScreen {
2     ...
3     private boolean simulationState;
4     private World physicsWorld;
5
6     public boolean isSimulationActive() {
7         return simulationState;
8     }
9
10    public void simulateWorld() {
11        // Invocación con parámetros requeridos para la simulación.
12        physicsWorld.Step(1.0f / 60.0f, 8, 3);
13    }
14    ...
15 }
```

*Código 23* – Simulación del mundo físico para el videojuego Círculos mediante el uso del objeto `World`. 
Incorporación de elementos al mundo físico

Una vez inicializado la simulación de la física, podemos agregar objetos con el objetivo de interactuar entre ellos, por ejemplo, siguiendo la temática del juego Círculos, incorporamos un número determinado de círculos. Para cumplir dicho objetivo, invocamos a los siguientes elementos heredados de la clase GamePlayScreen:

```java
1  class GamePlayScreenCircles : GamePlayScreen
2  {
3      ...
4      public GamePlayScreenCircles() {
5          ...
6              // Inicializa una lista de elementos del tipo GameElement.
7              initializePhysicsElements();
8          ...
9      }
10      ...
11  }
```

*Código 24 – Inicialización de elementos a ser incorporados al mundo físico.*

Nota:

Cabe destacar que el método initializePhysicsElements es heredado de la clase GamePlayScreen donde aloja una estructura de lista con elementos del tipo GameElement (para más información - 3.5.2 Representación física de la interacción entre elementos). A lo largo del capítulo utilizaremos métodos utilitarios obtenidos desde GamePlayScreen que permiten manipular estructuras, por ejemplo, del tipo Lista. Dicha propiedad permite obtener mayor lectura del código y evita la necesidad de declarar las mismas propiedades en diversas clases hijas.
Una vez inicializado la estructura de elementos, agregamos un número limitado de círculos:

```java
1  class GamePlayScreenCircles : GamePlayScreen
2  {
3      ...
4      public GamePlayScreenCircles(...) {
5          ...
6          // Inicializar una lista de elementos del tipo GameElement.
7          initializePhysicsElements();
8          // Agregar círculo #1.
9          addPhysicElement(new Circle(...));
10         // Agregar círculo #2.
11         addPhysicElement(new Circle(...));
12         ...
13         // Agregar círculo #N.
14         ...
15      }
16      ...
17  }
```

**Código 25 – Incorporación de elementos al mundo físico.**

**Nota:**
Como se vió en 3.5.3 Representación de elementos específicos del juego, la creación de un GameElement del tipo Circle requiere especificar propiedades como posición y radio.

Por último, para poder mostrar los círculos creados anteriormente, invocamos al método `drawElementsInTheWorld` de la clase GamePlayScreen de manera tal de representar todos aquellos objetos involucrados en el mundo físico. Dichos elementos, no requieren ser llamados dentro del ciclo de Update debido a que su actualización se realiza implícitamente dentro de la simulación física del juego:
Como se mencionó en la sección Representación física de la interacción entre elementos, cada objeto posee la característica de representarse por sí mismo, por lo tanto, sólo invocamos al método drawElementsInTheWorld para representar todos los elementos del mundo físico:

```csharp
class GameplayScreenCircles : GameplayScreen
{
    ...
    public override void Draw(...) {
        ...
        // Representar todos los elementos pertenecientes al mundo físico
        drawElementsInTheWorld();
    ...
}
```

**Código 26** – Representación de los elementos del mundo físico.

```csharp
class GameplayScreen : GameScreen
{
    private List<GameElement> elements;
    ...
    
    public void drawElementsInTheWorld() {
        for (int i = 0; i < elements.Count; i++) {
            // Dibujar elementos en pantalla considerando efectos particulares.
            elements[i].display(ScreenManager.SpriteBatch,
                                 ScreenManager.BasicEffect);
        }
}
```

**Código 27** – Delegación de la representación de cada elemento del tipo GameElement.
Los pasos detallados anteriormente se pueden ver en la siguiente figura:

![Figura 35 – Integración del esqueleto y los círculos.](image)

Validación de la lógica del videojuego

El siguiente paso a definir son las reglas específicas de validación del videojuego, que determinarán cuando el jugador gana o pierde. Todo esto se debe determinar dentro del método `Update`, en donde se llama al método `verifyGameStatus` para realizar los chequeos correspondientes y determinar cuándo se gana o se pierde el videojuego:
Clase `GamePlayScreenCircles`:

```java
class GamePlayScreenCircles : GamePlayScreen
{
...
public override void Update(...) {
...
verifyGameStatus();
...
}
```

*Código 28 – Validación del estado del videojuego Círculos.*

Este último está implementado dentro de la clase `GamePlayScreen`, con lo cual puede accederse desde cualquier instancia que herede de la misma:

```java
class GamePlayScreen : GameScreen
{
...
protected void verifyGameStatus() {
    int state = getPlayerState();
    // Si state es 1 el jugador ganó y muestra un mensaje.
    if (state == 1) {
        showWinnerMessage();
    } else {
        // Si state es -1 el jugador perdió y muestra un mensaje.
        if (state == -1) {
            showLooserMessage();
        }
    }
...
}
```

*Código 29 – Validación del estado del videojuego Círculos dentro de la clase `GamePlayScreen`.*

La validación del estado del juego (si pierde o gana) se determina de la misma manera para todos los videojuegos. La variante importante a resaltar se observa en el método abstracto `getPlayerState` de la clase `GamePlayScreen`. De esta manera, dado que cada videojuego posee una lógica distinta, cada instancia que extienda de `GamePlayScreen` debe implementar el método `getPlayerState`: 
Código 30 – Clase abstracta GamePlayScreen.

Código 31 – Implementación del método abstracto getPlayerState.

Como se mencionó anteriormente cada videojuego que extienda de GamePlayScreen debe implementar el método abstracto getPlayerState, de manera tal de determinar la lógica particular de dicho juego. Toda la funcionalidad o propiedades requeridas para llevar a cabo la validación debe realizarse (para el caso del videojuego Círculos) dentro de la clase GamePlayScreenCircles.

Haciendo una recopilación de todo lo incorporado al juego, se han considerado los siguientes puntos:

- visualización gráfica e interacción del jugador en el mundo físico del videojuego.
● visualización gráfica e interacción de los elementos del videojuego con el mundo físico.
● definición de las reglas propias del videojuego.
● determinar cuándo se ganó o se perdió el videojuego.

Con estos últimos métodos incorporados se finaliza con la implementación logrando el videojuego completo.

Configuración personalizada

Adicionalmente, podemos obtener propiedades particulares que permiten dar un valor agregado a la lógica de los videojuegos. Dichas propiedades son almacenadas dentro de un archivo de configuración `game.xml` que se muestra a continuación:

```xml
<game>
  <gamename>círculos</gamename>
  <levels>
    <level>  // Nivel 1.
      <gravity>0</gravity>  // Gravedad aplicada al mundo físico.
      <elements>10</elements>  // Cantidad de elementos.
      <time>50000</time>  // Tiempo disponible.
    </level>
    <level>  // Nivel 2.
      <gravity>0</gravity>
      <elements>20</elements>
      <time>30000</time>
    </level>
    <level>  // Nivel N.
      <gravity>0</gravity>
      <elements>20</elements>
      <time>30000</time>
    </level>
    ...
  </levels>
</game>
```

*Código 32 – Archivo de configuración para el videojuego Círculos.*

Dichas propiedades varían en base a las necesidades de cada videojuego, permitiendo agregar opciones para cada nivel de dificultad que se quiere considerar. Del mismo modo, podemos almacenar un conjunto de propiedades con sus valores, relacionados a un nivel en particular.
Por otro lado, para poder utilizar las propiedades de los niveles de cada juego, simplemente se invoca la clase DataManager que provee propiedades almacenadas en memoria, ya sea para videojuegos o para usuarios relacionados con la sesión actual.

```
1  ...
2  GamePlayScreenCircles circleGame = new GamePlayScreenCircles(
3  ScreenManager.getWidthScreen(),
4  ScreenManager.getHeightScreen(),
5  DataManager.Instance.getGames()["circulos"].getLevels());
6  ...
```

**Código 33 – Obtención de propiedades desde el archivo de configuración.**

**Videojuegos adicionales**

Con el objetivo de instanciar el framework se crearon dos videojuegos adicionales para mostrar cómo implementar diferentes ideas, modificando y configurando el menor número de recursos disponibles.

**Videojuego Fechas**

Para dicho juego se propuso una mecánica muy simple, que permite al jugador seleccionar las direcciones correctas de un conjunto de flechas que aparecen en pantalla cada cierto tiempo. Para lograr el objetivo planteado el jugador debe elegir con sus manos hacia qué dirección apunta una flecha en pantalla, las direcciones planteadas por defecto son: arriba, abajo, izquierda y derecha.
Así como en el videojuego anterior, para la implementación de dicho videojuego se creó una nueva clase denominada GamePlayScreenArrows. La misma hereda métodos que deben ser implementados para poder permitir el funcionamiento del juego.
Al igual que el videojuego anterior, algunos pasos en el desarrollo del mismo son similares, por lo cual no van a ser repetidos en detalle, sino que solamente son mencionados como parte de la misma implementación:

- Incorporación del jugador dentro del videojuego.
- Incorporación física y gráfica de los elementos específicos del videojuego.

Incorporación del jugador al videojuego

A diferencia del videojuego Círculos, por la mecánica de juego propuesta, en este videojuego no es necesaria la incorporación al mundo físico del esqueleto del jugador. Por lo tanto, solamente el cuerpo del jugador será visualizado en pantalla, pero no podrá interactuar físicamente con los elementos que están dentro del mundo del videojuego.
Para lograr que el esqueleto no esté incorporado dentro del mundo físico, debemos instanciar un objeto del tipo `Skeleton` mediante el constructor de la clase por defecto, es decir sin argumentos:

```java
class GamePlayScreenArrows : GameplayScreen
{
    public GamePlayScreenArrows () {
        skeleton = new Skeleton();
        ...
    }
    ...
}
```

**Código 34 – Creación del esqueleto del usuario para el videojuego Flechas.**

Validación de la lógica del videojuego

Del mismo modo que en el videojuego `Círculos`, se deben definir las reglas específicas del juego y determinar cuando el jugador gana o pierde. Para ello implementamos la lógica determinada en el método abstracto `getPlayerState`:

```java
class GamePlayScreenArrows : GameplayScreen {
    public int getPlayerState() {
        // Si no se terminaron las indicaciones del nivel, seguir.
        if (arrowsIndex != elements) {
            return 0;
        }
        // Si alcanzo el mínimo para pasar el nivel, ganó.
        if (elements == hits) {
            return 1;
        }
        // Se terminaron las indicaciones del nivel, perdió.
        return -1;
    }
    ...
}
```

**Código 35 – Determinación del estado actual del videojuego Flechas.**
En éste segmento se determina el estado del juego en un determinado tiempo. Los posibles estados del juego son:

- cuando el jugador pierde (-1),
- cuando el jugador alcanza a tocar todos los elementos mostrados en pantalla y gana (1),
- y por último cuando el jugador continúa en juego; siendo (0) el valor por defecto.

**Videojuego Seleccionador**

En este videojuego el jugador tiene como objetivo clasificar los círculos de colores verdes y amarillos. Con sus manos el jugador debe indicar a cuál de las cajas corresponde cada uno de los círculos que van apareciendo en pantalla cada cierto tiempo. Los colores de los círculos van variando de forma aleatoria, y en caso de ser necesario se puede especificar la secuencia de colores que se desea dentro del archivo de configuración XML (ver sección *Configuración personalizada*).

Así como en los videojuegos anteriores, para la implementación de dicho videojuego se creó una nueva clase denominada *GamePlayScreenChooser*. La misma hereda métodos que deben ser implementados para poder permitir el funcionamiento del juego.
Incorporación del jugador al videojuego

Del mismo modo que en el videojuego Flechas, no es necesaria la incorporación física del esqueleto del jugador.

Validación de la lógica del videojuego

Al igual que en los videojuegos anteriores, se deben definir las reglas específicas del juego y determinar cuando el jugador gana o pierde.

Como se puede ver en Código 36, se requieren dos métodos adicionales, controlZone y verifyElementBox, que son los encargados de verificar si la mano del jugador está dentro de la zona de juego indicada y chequear si el elemento de color se encuentra dentro de la caja de color correcta (ver Figura 40).
Código 36 – Actualización del estado del videojuego Flechas.

Figura 40 – Chequeo de zonas de juego realizado por métodos adicionales

Para realizar la lógica de validación del videojuego, implementamos el método abstracto `getPlayerState`:
class GameplayScreenChooser : GameplayScreen
{

private int getPlayerState()
{
  int state = 0; // Por default se considera que sigue jugando.
  // Gana si ya se clasificaron todos los elementos y no hubo
  // ningun error.
  if (currentElement > totalElements && faults == 0) {
    state = 1;
  }
  // Pierde si ya se clasificaron todos los elementos y hubo algún error.
  else if (currentElement > totalElements && faults > 0) {
    state = -1;
  }
  // Pierde si se quedó sin tiempo de juego.
  else if ((timeSpan.Minutes == TimeSpan.Zero.Minutes &&
             timeSpan.Seconds == TimeSpan.Zero.Seconds) {
    state = -1;
  }
  return state;
}
...
Capítulo 5

Conclusiones

En el presente trabajo se presentó una plataforma que permite el desarrollo de juegos integrando diferentes tecnologías, tales como Kinect, XNA y Box2D. La misma permite fundamentalmente desarrollar videojuegos destinados a educación de niños que posean algún tipo de limitación en sus habilidades. La plataforma le permite al desarrollador ahorrar tiempo en la integración de las diferentes tecnologías y le permite centrarse directamente en el desarrollo propio del videojuego en cuestión.

Entre otras ventajas de la plataforma, se puede mencionar que todos los juegos que fueron desarrollados están dentro de una misma aplicación a la que se la denominó SIMON. La misma tiene un sistema de menú por el cual se puede acceder a todos los juegos desarrollados y perfiles de los niños que son usuarios de la aplicación.

También se debe destacar el sistema de perfiles de usuario que se incorporó, cada niño tiene uno en donde se guarda su foto y las estadísticas de cada uno de los juegos que fue utilizando. Esto sirve para que los educadores puedan tener un seguimiento de la cantidad de veces jugadas, los niveles jugados, tiempos, aciertos y errores cometidos. Además, cada uno de los juegos es personalizable y serializable mediante archivos de configuración, por el cual se pueden modificar el tiempo de juego, la cantidad de objetos que intervienen, la velocidad en la que aparecen, etc.

El uso de la plataforma fue probado en el desarrollo de un conjunto de juegos destinados a la Asociación Tandilense de Ayuda al Discapacitado (ATAD), dentro del proyecto "Aplicaciones interactivas para el desarrollo motor y cognitivo de niños y jóvenes con discapacidad en espacios educativos", a través del subsidio obtenido para Proyectos de Tecnologías para la Inclusión Social, en el marco del Programa Consejo de la Demanda de Actores Sociales (PROCODAS).

Los diferentes juegos que se han mencionado dentro de la sección de Instanciación, fueron desarrollados luego de varias entrevistas realizadas con el personal de ATAD donde se obtuvieron los distintos requerimientos a tener en cuenta a la hora de desarrollar los mismos. Gracias a la plataforma se logró desarrollar distintos videojuegos de una forma más ágil en comparación a realizarlos sin la utilización de la misma. Los puntos más importantes a tener en cuenta a la hora de desarrollar un juego nuevo es implementar los métodos principales que permiten la representación de los elementos, actualización de estados de los objetos, chequeos de condiciones y otros aspectos propios del juego en sí mismo. Por otro lado, la misma plataforma se ocupa de representar el esqueleto del
jugador, de la inicialización del motor físico y, por último, del manejo de estadísticas y administración de los perfiles de usuarios.

Finalmente, la integración de dispositivos como Kinect potencia la creación de diversas temáticas de videojuegos relacionados a motricidad. Por el contrario, la utilización de dicho dispositivo en una etapa beta de implementación dificultó en menor medida la integración con la plataforma.

**Trabajos Futuros**

A lo largo del desarrollo de la plataforma han surgido algunas ideas de posibles mejoras que se pueden introducir a la misma y que pueden ser el comienzo de futuros trabajos. Entre ellos se pueden mencionar:

- Mejorar la captura de estadísticas de los juegos, para luego generar mejores reportes que puedan ser utilizados por los educadores.
- Incorporar una base de datos más robusta para la captura de estadísticas de los jugadores en cada uno de los juegos de la plataforma.
- Es posible mejorar el manejo de perfiles de los niños que utilizan los juegos y se pueden agregar nuevas características que permitan configurar automáticamente los parámetros de cada uno de los juegos.
- Realización de una extensión de la plataforma para dar soporte a la nueva versión de Kinect, ya que la utilizada en el presente trabajo es la primera generación de Kinect.
- Actualmente el presente trabajo está desarrollado con el SDK 1.7. Se puede realizar una actualización de la versión del SDK, en función de proveer nuevas funcionalidades a la plataforma como el mapeo de caras, reconocimiento de nuevos gestos y otras más.
- A partir del SDK 1.8 es posible desarrollar aplicaciones en JavaScript, por lo tanto, se podría llegar a pensar en una versión de la aplicación que sea capaz de ejecutarse en la ventana de un navegador.
- Integración con múltiples dispositivos adicionales, como por ejemplo Leap Motion, Wiimote entre otros; que permiten ampliar el abanico de posibilidades de creación de videojuegos.

Como dato final, podemos citar que el presente trabajo ha originado líneas de aplicación complementarias en otros trabajos de tesis. Actualmente se está trabajando en mejoras de usabilidad de SIMON mediante una herramienta destinada a los educadores. La misma permite la personalización de los juegos a través de un editor, posibilitando cambiar
parámetros de cada uno de los juegos ajustándose a las necesidades de cada uno de los niños que los usan. Adicionalmente, se tienen en cuenta apreciaciones de los docentes en distintas reuniones que se establecen mediante el uso de los videojuegos, haciendo que cada uno sea más personalizado y particular.
Bibliografía

Richard Harper, Tom Rodden, Yvonne Rogers and Abigail Sellen - Microsoft Research Ltd. - 2008

Joseph J. LaViola Jr. – 2009

Thomas Perl, Benjamin Venditti and Hannes Kaufmann - Interactive Media Systems Group University of Technology Vienna - 2012
http://publik.tuwien.ac.at/files/PubDat_218820.pdf

[4] Report on the educational use of games: An exploration by TEEM of the contribution which games can make to the education process
McFarlane, A., Sparrowhawk, A. & Heald, Y. - 2002
http://nexus.bremen.de/Library.nsf/bf25ab0f47ba5dd8525649006b15a4/be501a19594142bac125736e00588a6d/$FILE/Games_in_Education_TEEM_gamesined_full.pdf

Cesarone, B. ERIC Digest ED446825 - 2000

[6] The role of video game playing in adolescent life: is there reason to be concerned?

[7] Proyecto Insight in Kerala, India
http://www.itmission.kerala.gov.in/insight.php

[8] The Potential of Kinect in Education

[9] Exerbraining for schools: Combining body and brain Training


N.A.R. Ayala - Procedia Computer Science - 2013

[12] Natural user interfaces to teach math on higher education
M. A. M. Echeverría - Procedia Social and Behavioral Sciences, vol. 106 - 2013
[13] Video game-based exercise, Latino children’s physical health and academic achievement
Z. Gao - American Journal of Preventive Medicine, vol. 44 - 2013

[14] Improving Learning Performance with Happiness by Interactive Scenarios


[16] Beginning XNA 3.0 Game Programming From Novice to Professional
Alexandre Santos Lobão, Bruno Evangelista, José Antonio Leal de Farias, Riemer Grootjans. Apress 2009

[17] El Patrón Singleton