Problemas matemáticos que surgen de la programación de videojuegos y su relación con el desarrollo del pensamiento matemático

Rafael Miranda Molina[1]

RESUMEN

En el contexto del Penta UC, un programa de enriquecimiento extracurricular orientado a la educación de niños y jóvenes con talento académico, se presenta una categorización de problemas matemáticos que surgen en un curso de programación de videojuegos. A partir de su categorización, se analiza la relaciona entre estos problemas y algunos estándares nacionales e internacionales de educación matemática, para delinear su potencial para el desarrollo del pensamiento matemático.

Palabras clave: videojuego, programación, pensamiento matemático, problema.

Mathematical problems arising from the game programming and its relation to the development of mathematical thinking

ABSTRACT

Categorization of mathematical problems that arise in a game programming course, is presented in the context of Penta UC, and extracurricular enrichment program aimed at educating children and youth with academic talent. From their categorization, its relationship with some national and international standards of mathematics education is analyzed, to delineate its potential for the development of mathematical thinking.

Keywords: videogame, programming, mathematical thinking, problem.

Fecha de recepción: 01 de julio de 2014

Fecha de aceptación: 18 de agosto de 2014

[1]Profesor de matemática e informática educativa (UMCE), Magister en educación matemática (UFRO). Docente del Programa Penta UC de la Pontificia Universidad Católica y Coordinador de Nivelación del Programa de Acceso Inclusivo, Equidad y Permanencia de la Universidad de Santiago. Este artículo corresponde a una síntesis de la tesis de magister "Potencial de la programación de videojuegos para el desarrollo del pensamiento matemático de estudiantes en edad escolar", de Octubre 2012. Correo electrónico: rafael.miranda@usach.cl

1. ANTECEDENTES

El presente trabajo se desarrolla en el contexto del Programa para Niños y Jóvenes con Talentos Académicos de la Pontificia Universidad Católica de Chile, PENTA UC. Desde el año 2007 este programa ha incluido en su programación académica el curso "Jugando con la programación", un curso de programación de videojuegos, en el que se puede observar una alta presencia de problemas matemáticos, y luego, es interesante relacionarlo con el desarrollo del pensamiento matemático.

"Jugando con la programación" es una propuesta didáctica que surge de la necesidad de acercar la programación de computadores a los intereses de los estudiantes del segundo ciclo básico, sin considerar prerrequisito alguno (informático o matemático) para su participación. Esta condición es necesaria dado que en niveles superiores los estudiantes podrán optar por otros cursos de programación en los que se estudia la programación estructurada más formalmente. De esta manera, la elección de centrar este curso de programación en los videojuegos, cumple una doble función: se trata de programas con una fuerte componente gráfica y al mismo tiempo se alinea con la motivación de los niños de hoy, ambos elementos facilitadores de aprendizajes.

Una descripción usual de un videojuego, es la de "un juego electrónico que involucra la interacción con un usuario para generar una respuesta gráfica" (Barrionuevo, 2009). Desde esta lógica, muchos videojuegos pueden entenderse esencialmente como un conjunto de animaciones controladas por ciertas condiciones, ya sea, acciones del usuario o reglas subyacentes. Esta conceptualización deja en evidencia que los principales temas matemáticos involucrados en su programación, son aquellos que permiten describir el movimiento de objetos, y los métodos que por excelencia se utilizan en los entornos de programación son los analíticos. En consecuencia, la programación de un videojuego es en gran medida una tarea que involucra a la geometría analítica y transformacional (Begel, 2004; Kodicek, 2005), además de la programación propiamente tal.

Dado que este curso responde a las necesidades de los estudiantes, las necesidades del programa y a este particular contexto, se trata de una propuesta original, que no sigue ninguna secuencia didáctica convencional que pudiera haberse incorporado de algún autor o experiencia previa. La innovación que significó este curso está en proponerse como una respuesta a la problemática de cómo enseñarle a programar a niños talentosos, sin abordar los contenidos usuales de la programación estructurada, y además hacerlo de manera entretenida y desafiante para ellos.

Para hacerle justicia a tan ambiciosos objetivos, "Jugando con la programación" en las 15 versiones que se ha dictado, sus actividades y contenidos se fueron ajustando semestre a semestre, especialmente debido a la aparición de diversos desafíos asociados a contenidos del currículum de matemática, como el cálculo de la pendiente entre dos puntos, coordenadas polares e incluso

lógica, entre otros. En general todos los estudiantes tuvieron múltiples oportunidades para elegir qué tipo de juegos programar, y ante la multiplicidad de intereses y de juegos que previamente habían jugado, van apareciendo con el tiempo diversas situaciones que no estaban consideradas en el diseño inicial del curso. Entonces, a partir de las propuestas de los estudiantes, la cantidad de posibilidades se fue ampliando.

A pesar de que los contenidos del curso no fueron explícitamente matemáticos, diversos desafíos de este tipo surgieron naturalmente como consecuencia de los tipos de juegos que fueron proponiendo los mismos niños. Dado que en este entorno, asociado al lenguaje KPL (acrónimo de *Kids Programming Language*), está referido a la programación estructurada y hace uso de coordenadas en el plano cartesiano; muchas veces nos encontramos con problemas muy complejos para niños de 6º básico. Algunos de estos problemas fueron facilitados o incluso resueltos por los profesores; otros, más simples, fueron solucionados por los estudiantes, aunque sin centrarnos en lo matemático, sino en el juego a programar.

Esto deja en evidencia, el hecho de que a los diversos juegos que se programaron, subyacen problemas matemáticos que son interesantes para estudiantes de otros niveles educacionales, y que por no ser contenidos explícitos del curso, pasaron más bien inadvertidos para los estudiantes. Muchos de ellos utilizaron ideas matemáticas que no se explicitaron o formalizaron, quedando estos problemas matemáticos y sus soluciones, ocultos en los códigos fuente de sus juegos. Dado que cada estudiante entregó cerca de 10 trabajos en cada semestre, la cantidad de trabajos analizados es cercana a 1.800.

En consecuencia, la problemática principal de este trabajo, se centra en todas aquellas situaciones matemáticas interesantes que surgieron de la programación de videojuegos y su relación con el aprendizaje matemático. Así, el propósito fundamental de esta investigación es el develar el potencial de la programación de juegos para el desarrollo del pensamiento matemático de niños y jóvenes en edad escolar.

2. PROBLEMA Y PREGUNTAS DE INVESTIGACIÓN

El área problemática general en la que se desarrolla este trabajo, es en la relación entre la programación de videojuegos y el desarrollo del pensamiento matemático de estudiantes en edad escolar. Sin embargo, este estudio en ningún caso busca describir por completo tal relación, sino que más bien se central en la situación particular del curso en cuestión que se ha dictado desde el año 2006 en 15 ocasiones, sumando semestres regulares y semestres de verano.

Así, es importante tener en consideración los características particulares de esta experiencia como el haber trabajado con niños con talento académico de 6º a 8º básico, con el lenguaje KPL y considerando que el estudio de las estructuras clásicas de programación es contenido de cursos posteriores. Aun así, por particular que es esta propuesta, el potencial de la programación de

videojuegos se puede abordar tanto de la perspectiva de qué y cuánto aprendieron los estudiantes de este curso; así como también, qué más podrían aprender otros estudiantes al intentar programar los mismos tipos de videojuegos.

Para orientar el estudio se trabajó a partir de los siguientes problemas y preguntas:

Problema

- ¿Qué relación existe entre la programación de videojuegos y el desarrollo de pensamiento matemático en estudiantes de edad escolar?

Preguntas de investigación

- ¿Con qué tipo de problemas matemáticos se relacionan los videojuegos programados?
- ¿Qué relación tienen los problemas trabajados por ellos con el desarrollo del pensamiento matemático?

3. DESCRIPCIÓN DE CURSO

"Jugando con la programación", es un curso que se implementa por primera vez en la temporada de verano 2007 del Penta UC. A diferencia de los semestres regulares, en los que los cursos cuentan con 14 ó 15 sesiones de 3 horas, en la temporada de verano los cursos se organizan en 9 ó 10 sesiones de 5 horas.

Ante la necesidad de diseñar un curso de programación para enseñanza básica, que apelara a los intereses de niños y jóvenes, la decisión de que tratara de videojuegos era bastante natural. Sin embargo, era también necesario que no se incluyeran contenidos de los otros cursos de programación, que se dictaban para niveles superiores. Así, el enfoque que debía buscarse era más bien intuitivo.

El lenguaje de programación seleccionado fue KPL, Kids Programming Languaje, un lenguaje creado en 2005 con el propósito de acercar la programación de manera motivante a programadores novatos e intermedios (Schwartz, 2005). Se trata de un lenguaje que tiene ciertas similitudes con Visual Basic, y que admite elementos complejos de programación como métodos y estructuras, aunque como lo expresan sus creadores, orientado a la enseñanza.

La primera versión del curso fue una experiencia más bien exploratoria, a partir de la cual se observan diversas situaciones que permiten construir una secuencia didáctica más apropiada para niños de tales niveles (6º a 8º básico). Así, se observó que la mejor forma de describir un videojuego, era como un conjunto de animaciones que dependían de ciertas condiciones, como las teclas que el usuario presiona, la interacción con otros elementos o reglas predefinidas. Además, este tipo de animaciones dependen en gran medida de coordenadas y vectores, de manera que tales ideas matemáticas deben ser abordadas de alguna forma durante el curso.

Las unidades que se articulan desde la 2ª versión son:

- Dibujos en el plano cartesiano.
- Animaciones de imágenes.
- Control de animaciones.

Más allá de la tercera unidad, la secuencia didáctica continúa, aunque una vez que se logra controlar una determinada animación con el

teclado ya se cuenta con un esquema básico para un juego, por lo que esta secuencia permite conectar ideas intuitivas como las de dibujos en un plano cartesiano con el propósito de diseñar un videojuego. Además, es usual la implementación de proyectos de programación propios a partir de este punto, por lo que los temas a desarrollar dependen de los tipos de juegos que decidan crear los estudiantes.

3.1 OBJETIVOS Y CONTENIDOS DEL CURSO

Objetivo General

- Resolver problemas asociados a la funcionalidad e interactividad de juegos, diseñando, probando e implementando soluciones algorítmicas en el computador.

Objetivos Específicos

- Programar dibujos con formas geométricas en función de las coordenadas de vértices, centros y dimensiones de figuras.
- Programar animaciones a partir de la iteración de movimientos rígidos y controlarlas a partir de condiciones propias de juegos.
- Describir las condiciones necesarias para resolver problemas que involucran algoritmos secuenciales y condicionales.

Contenidos

Unidad 1: Dibujos

- Sistema de coordenadas. Dibujo de puntos y líneas
- Estructura básica de un programa
- Dibujos con figuras (elipses, círculos y rectángulos)
- Atributos de los dibujos (color, grosor de línea, color de fondo, etc.)
- Dibujos con el uso de Traslaciones
- Sentencia repetitiva simple (loop)
- Animaciones basadas en dibujos

Unidad 2: Animaciones

- Carga y visibilidad de Imágenes
- Traslación y rotación de imágenes
- Transformación, superposición y opacidad de imágenes (flip, scale, etc.)
- Continuidad del movimiento (traslación y rotación) de imágenes
- Movimientos simultáneos de imágenes y animación gif

Unidad 3: Juegos

- Animaciones con condiciones:
  - Rebote de una imagen en pantalla
  - Mover imagen con el teclado
- Estructura básica de un juego
  - Definición de variables
  - Animaciones con variables y fórmulas
- Sentencia condicional asociada a
  - Detección de teclas presionadas
  - Intersección de imágenes

3.2 NIVELES DE APRENDIZAJE

Si bien las actividades del curso fueron cambiando de una versión a otra, la estructura que se terminó aplicando en las últimas comprende una serie de hitos, a partir de los cuales se construyen los aprendizajes posteriores. Para dar cuenta de tales niveles, se describe a continuación la progresión de aprendizajes que se terminó de construir en las últimas versiones del curso:

Nivel 1: Dibujos en el plano cartesiano (4 horas)

En las primeras clases, aun sin computadores, se estudian los comandos más básicos de dibujo, que permiten generar diseños haciendo referencia a las coordenadas de los extremos de los segmentos que las componen.

Aprendizaje esperado

Describen las instrucciones que permiten construir un dibujo de KPL basado en segmentos.

Nivel 2: Dibujos con figuras (2 horas)

Usualmente aun en sala sin computadores, se practican comandos que permiten dibujar rectángulos, círculos y elipses, indicando las coordenadas de los centros y sus dimensiones. Posteriormente en el laboratorio de computación, aprenderán a escribirlos y ejecutarlos en el programa KPL, donde podrán comprobar si tales códigos generan los dibujos que esperaban. También se introducen algunos atributos de los dibujos, como colores, rellenos y grosor de línea.

Aprendizaje esperado

Describen y ejecutan las instrucciones que permiten construir un dibujo de KPL basado en segmentos, rectángulos, elipses y círculos.

Nivel 3: Dibujos iterativos (4 horas)

En esta actividad se introducen las traslaciones y una sentencia iterativa simple (loop), con la que se busca que generen diseños que involucran secuencias de figuras, mediante la iteración de traslaciones en el plano cartesiano.

Aprendizaje esperado

Construyen diseños basados en la iteración de transformaciones isométricas de líneas y figuras.

Nivel 4: Animaciones con dibujos (4 horas)

Las iteraciones antes trabajadas, pueden modificarse para generar movimiento, mediante el uso de pausas en los dibujos y borrando la pantalla. Así, en vez de generar un conjunto de figuras que siempre están visibles, en cada iteración se borra la figura antes generada, quedando siempre visible sólo una figura, lo que genera la ilusión de movimiento.

Aprendizaje esperado

Construyen animaciones basadas en la iteración de transformaciones isométricas de líneas y figuras.

Nivel 5: Animaciones de imágenes (5 horas)

En esta actividad se introducen los comandos que permiten incluir imágenes en pantalla (por ejemplo, descargadas de Internet). Mediante comandos similares a los que utilizaron previamente para dibujar, ahora pueden modificar la posición de las imágenes, lo que les permitirá moverlas en pantalla.

Aprendizaje esperado

Construyen animaciones basadas la aplicación de transformaciones isométricas a imágenes.

Nivel 6: Animaciones fluidas de imágenes (6 horas)

El problema de la fluidez o continuidad del movimiento se estudia en esta actividad, extendiendo las ideas de dibujos iterativos al movimiento de imágenes, de manera que, encontrando un equilibrio entre cuánto avanza en cada paso y la pausas intermedias, puede generarse un movimiento suficientemente fluido para que parezca continuo.

Además se desarrollan desafíos en los que una imagen debe moverse por un trayecto

previamente determinado, lo que involucra determinar el vector de traslación apropiado para que la traslación parta y termine donde corresponde. También se introducen otras transformaciones como la rotación y reflexión.

Aprendizaje esperado

Construyen animaciones fluidas, basadas en la iteración de transformaciones isométricas.

Actividad 7: Prototipo inicial de juego (6 horas)

Una vez que son capaces de generar diversos tipos de animaciones de imágenes, basadas en transformaciones isométricas, el paso siguiente para construir juegos es poder controlarlas con el teclado. En esta actividad el desafío principal es controlar la posición de una imagen, dependiendo de las teclas presionadas en el teclado. A este tipo de programa usualmente le denominamos un prototipo de juego, pues a partir de tal maqueta es posible armar muchos otros tipos de juegos.

Aprendizaje esperado

Construyen un programa en el que se puede controlar la posición de una imagen, en cuatro direcciones, con el teclado.

Nivel 8: Prototipos de juegos (2 horas)

Siguiendo la idea anterior, existen muchos juegos que requieren de otros esquemas de movimiento, de manera que en esta actividad se exploran esas otras posibilidades, en el sentido de controlar el movimiento de imágenes en pantalla de otras formas, por ejemplo, moviendo sólo verticalmente una imagen en cada borde de la pantalla, o controlar una imagen con cuatro teclas del teclado y otra imagen simultáneamente con otras cuatro teclas.

Aprendizaje esperado

Construyen programas en los que se pueden controlar imágenes de diversas formas, según las reglas de los juegos a programar.

Nivel 9: Tareas comunes de juegos (proyecto de programación, 8 horas)

Finalmente la construcción de un juego requiere implementar diversas características adicionales a los prototipos, las que dependen de los intereses de los estudiantes. Esta actividad usualmente se trabajó en grupos, dándoles apoyo personalizado según las tareas de programación que surgen de sus proyectos.

Aprendizaje esperado

Construyen videojuegos en los que se cuenta con reglas bien definidas para jugar y ganar o perder.

3.3 ASPECTOS CUALITATIVOS

Cabe destacar que todas estas acciones, como son desarrolladas en un entorno de programación, esencialmente corresponden a escribir códigos que el compilador (KPL) interpretará y ejecutará. Por ejemplo, en las primeras clases se estudiaron códigos que permiten dibujar punto a punto, como el siguiente:

Ejemplo de código de dibujo (clase 1)

moveTo(100,0) moveTo(100,100) moveTo(0,100) moveTo(0,0)	Cada línea de este código mueve el cursor a la coordenada indicada, dibujándose los segmentos que forman un cuadrado de lado 100 (pixeles).

Códigos de esta naturaleza son los que los estudiantes aprenden a leer, modificar y escribir, los cuales clase a clase van aumentando en complejidad, para describir soluciones algorítmicas a problemas esencialmente gráficos.

Ahora bien, para motivar a los estudiantes, el curso contó con una segunda parte en la que se organizaron proyectos de programación, donde los estudiantes pudieron programar los videojuegos que quisieron, aprovechando todo lo aprendido previamente. Durante tal etapa, surgen muchas ideas de videojuegos que no estaban consideradas en el diseño inicial, y por tanto, aparecen muchos problemas matemáticos emergentes.

Por ejemplo, en la tercera y cuarta versiones surge el interés de desarrollar juegos asociados al ping-pong, lo que involucra describir el rebote de una pelota en pantalla; o bien, en la quinta versión surge el interés de realizar juegos en los que se controla una imagen de una forma más similar a cómo de conduce un auto (controlando el giro y el avance de manera independiente), lo que genera la necesidad de ideas asociadas a las coordenadas polares.

Esta decisión de diseño del curso, de admitir que los trabajos a desarrollar en la segunda etapa provengan de los intereses de los estudiantes fomentó que surgieran una gran cantidad de problemas nuevos, los que permitió enriquecer el curso semestre a semestre y es la principal motivación para este estudio.

Otra decisión fundamental, fue la de utilizar una metodología más bien personalizada, condición a veces compleja para cursos con más de 20 estudiantes muy activos y que se interesaban frecuentemente por problemas complejos de programación. Como consecuencia de esto, el trabajo con los estudiantes fue de un diálogo permanente en torno a los problemas involucrados, y es en este diálogo que se da una interesante conexión entre dos lenguajes: el lenguaje natural, referido a los juegos, y el lenguaje matemático aplicado a la programación. Así, frecuentemente estudiantes preguntaban sobre problemas que se les presentaban, haciendo referencia a la metáfora del juego, por ejemplo, simular el rebote de una pelota con los bordes de la pantalla a veces preguntaban "*¿cómo le pongo murallas al juego?*". Tal lenguaje en el que los estudiantes hacían sus preguntas, en adelante, se describirá como la "metáfora" de un juego, que usualmente involucra menciones a "naves", "misiles", "disparos", "choques", "rebotes", etc., pero que en realidad se refieren a condiciones matemáticas.

Es razonable entonces preguntarse qué tipo de problemas se están presentando en estos entornos, especialmente de aquellos problemas emergentes que surgen de los intereses de los niños y jóvenes, como también de las dificultades

con las que se encontraron. Ninguno de estos aparece como en los textos de estudio, formulado en un enunciado, sino más bien ocurren de manera gráfica, cuando el estudiante nota que algo no funciona como esperaba.

4 EXTRACCIÓN DE PROBLEMAS Y NATURALEZA DE LOS PROBLEMAS DETECTADOS

Hasta ahora se ha hablado de manera genérica de problemas matemáticos involucrados en la programación de videojuegos, sin embargo, los objetivos propuestos para este trabajo requieren de identificarlos y categorizarlos, de manera que es necesario adoptar un cierto lenguaje para comunicar con mayor precisión tales situaciones. Dada la diversidad de acepciones distintas que pueden encontrarse en la literatura para el término "problema", es necesario aclarar algunos aspectos sobre la naturaleza de los aquellos involucrados en este curso. Para lo que se tomará como base la caracterización de Raffaella Borasi (Alfaro, 2008), que hace referencia a sus elementos esenciales.

La presencia de problemas matemáticos es consecuencia de la necesidad de describir en un determinado lenguaje de programación, condiciones matemáticas que frecuentemente involucran coordenadas y vectores. En gran medida la relación entre estos entornos de programación y la matemática, es a través de la geometría transformacional para describir movimientos. Tómese el siguiente código como ejemplo:

```
MoveSpriteBy("nave",x,y)
If GetSpriteLeft("nave")
> 640 Then      x = - x
End If
If GetSpriteLeft("nave")
< 0 Then        x= - x
End If
If GetSpritetop("nave") >
480 Then        y= - y
End If
If GetSpritetop("nave") <
0 Then  y= - y
End If
```

En esta parte del programa, una imagen llamada "*nave*", que se traslada según el vector *(x,y)*.
Los comandos "*getSpriteLeft"* y "*getSpriteTop"*, entregan sus distancias a los bordes izquierdo y superior de la pantalla, respectivamente.
Luego, si la imagen está a más de 640 ó menos de 0 pixeles del borde izquierdo, la componente horizontal del vector de traslación cambia de signo.
Análogamente si la imagen está a más de 480 ó menos de 0 pixeles del borde superior cambia el signo de su componente vertical.

Analizando estos códigos, podemos extraer un problema matemático que se puede formular como: simular el rebote de una imagen con los bordes de la pantalla. Es importante notar que tal problema no está formulado, sino que lo estamos identificando a partir de la solución que este algoritmo describe. A tal proceso denominaremos *extracción* de un problema desde el código fuente de un videojuego.

Ahora bien, la caracterización de Borasi es útil para identificar los aspectos clave de distintos tipos de problemas matemáticos, sin embargo, determinar qué tipos de problemas estamos identificando requiere de contextualizarlos. El problema del rebote de una imagen, por ejemplo, surge frecuentemente a partir de los juegos elegidos por los estudiantes para programar. Pero la forma como surgen, es con una formulación centrada en la metáfora del juego, es decir, la forma como los estudiantes plantean tal problema

es con preguntas como; "*¿Cómo agrego murallas?*" o bien "*¿Cómo hago que rebote la pelota?*". Tal forma de describir el problema no es útil para formular un plan[2]; puesto que no hace referencia a conceptos matemáticos o de programación, de manera que como parte de comprender el problema, es necesario orientar a los estudiantes para formularlo de manera que quede en evidencia qué quiere decir, matemáticamente, que la pelota rebote.

En consecuencia, estos problemas surgen de manera natural, su formulación está en gran medida implícita y su reformulación es parte la etapa inicial de *"comprender el problema"*. Es por tal razón, que para este análisis se entenderá como problema, lo que Borasi denomina ya sea *"situaciones problemáticas"* o *"situaciones"*, cuyos elementos caracteriza de la siguiente forma (Borasi, 1986 en Alfaro, 2008):

Tipo	Contexto	Formulación	Soluciones	Método
Situación problemática	Sólo parcial en el texto	Implícita, se sugieren varias, problemática	Varias. Puede darse una explícita	Exploración del contexto, reformulación, plantear el problema.
Situación	Sólo parcial en el texto	Inexistente, ni siquiera implícita	Creación del problema	Formulación del problema.

En algunos casos, como el ejemplo anterior, la formulación no existe y en gran medida se debe partir por formular el problema. Desde tal perspectiva, el problema del rebote es más bien una situación. Sin embargo una vez formulado el problema, es necesario lograr que la pelota efectivamente rebote, problema que podría catalogarse de situación problemática.

Luego, podría entenderse que algunos problemas son situaciones y otras situaciones problemáticas, lo importante de aclarar es que la formulación es usualmente el primer paso para comprender el problema, y difícilmente surgirán en este contexto problemas ya formulados; salvo por aquellos propuestos en guías de ejercicios.

5. EXTRACCIÓN DE PROBLEMAS

Para crear la lista inicial de problemas, se revisaron cerca de 1800 códigos fuente, correspondientes a los principales trabajos de todos los estudiantes de las versiones 2 a 8, lo que corresponde a cerca de 10 trabajos en promedio por estudiante. Para la selección de trabajos, se utilizaron métodos automáticos para descartar duplicados, seleccionando la versión más reciente cuando existieran distintas versiones de un mismo archivo.

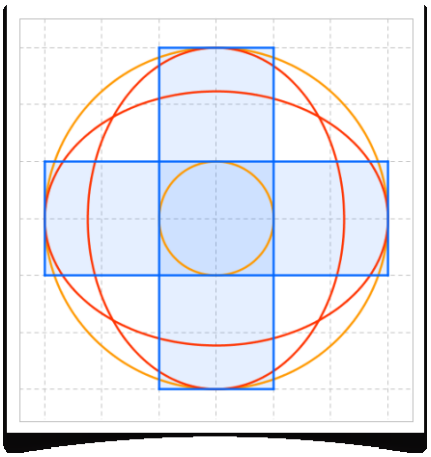
De esta forma, se identifican 30 tipos de problemas distintos, en cuya formulación se hace, a veces, referencia a elementos de la metáfora del juego del que surgen, como "disparar misiles" por ejemplo.

[2] Nótese que se está haciendo referencia al modelo de Polya, donde la primera fase, de comprensión del problema, requiere de formularlo (Polya, 1985).

5.1 CATEGORIZACIÓN Y ANÁLISIS DE PROBLEMAS

Tomando en consideración estas diferencias en cuanto a los elementos involucrados en su solución, más que su formulación, se organizan en la siguiente lista de 6 categorías.

Categoría 1: Dibujos en el plano cartesiano



- Dado un dibujo, replicarlo en el plano cartesiano, a partir de las coordenadas de los vértices.
- Dado un dibujo, replicarlo en el plano cartesiano, a partir de la posición y dimensiones de rectángulos, círculos y elipses que lo componen.
- Construir diseños geométricos usando iteraciones de transformaciones geométricas.
- Dibujar una curva, usando sólo líneas rectas.
- Construir animaciones con figuras geométricas.

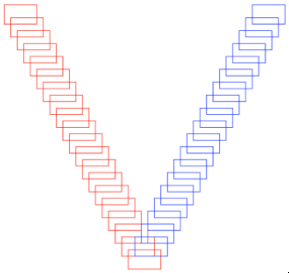
Este primer grupo de problemas que se presentan, consiste en aquellos que involucran dibujos en el plano cartesiano.

Esta es la primera instancia de familiarización con el entorno de programación, utilizando comandos de dibujo que hacen referencia a coordenadas. Así, se propone replicar dibujos dados, lo que requiere de identificar las coordenadas de los extremos de los segmentos que los conforman, como también hacer referencia a las dimensiones y posiciones de círculos, elipses y rectángulos.

Esto da origen a múltiples situaciones matemáticas interesantes, por ejemplo el establecer qué tipo de figuras se tienen en un dibujo dado, identificar condiciones que permitan replicarlos, relacionar dimensiones (radios, medidas de los lados de rectángulos y ejes de elipses) y posiciones (centros, vértices, etc.). También surgen conceptos que son comunes a la programación y la matemática, como es la eficiencia (o si se quiere, elegancia) de una determinada solución, planteándoles frecuentemente cómo sería posible reducir la cantidad de instrucciones para obtener el mismo resultado visual.

La idea de la eficiencia de una solución es el conector más natural para el paso siguiente, que es el automatizar la creación de dibujos con el uso de sentencias iterativas. Así, es posible crear un diseños que involucran muchos objetos, con sólo unas pocas líneas de código, lo que requiere de describir cómo iterar creación de figuras y transformaciones.

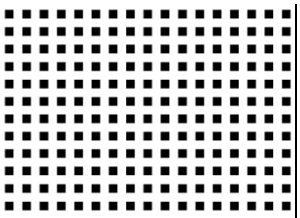
Ejemplo de dibujo iterativo



```
Loop 20
        rectangle(30,10,fals
e)
        moveby(5,10)
End loop
Loop 20
        rectangle(30,10,fals
e
        moveby(5,-10)
End loop
```

Además de los aspectos geométricos de estos diseños, es interesante también la combinación con aspectos algorítmicos, como la necesidad de anidar iteraciones, para ciertos diseños. Tómese como ejemplo de esto un diseño análogo a un tablero de ajedrez, que requiere de considerar iteraciones para disponer cuadrados hacia la derecha, y luego repetir cada fila hacia abajo.

Ejemplo de dibujo con doble iteración



```
Loop 12
        Loop 17
                rectangle(10,10,tru
e)
                moveby(20,0)
        End loop
        moveby(-340,20)
End loop
```

La lógica que involucran las animaciones y dibujos iterativos es esencialmente la misma. La diferencia entre el movimiento horizontal de un cuadrado, y el dibujo de una serie de cuadrados organizados horizontalmente, es esencialmente la cantidad de cuadrados visibles. De tal forma, la animación de un dibujo requiere que justo antes de agregar un nuevo objeto se borre lo anterior. Con esta lógica es que usualmente se hacía la transición de dibujos iterativos a animaciones en el curso, donde la variable nueva en este último caso será el tiempo.

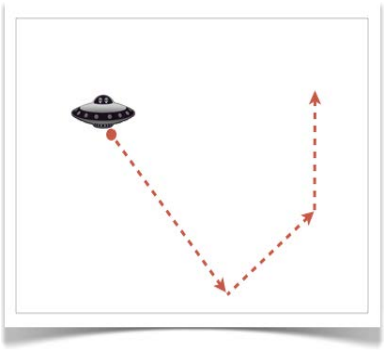
Ejemplo de dibujo y animación equivalentes

```
Loop 12
        rectangle(10,10,
true)
        moveby(20,0)
End loop
```

```
Loop 12
        clear()
        rectangle(10,10,t
rue)    moveby(20,0)
        delay(20)
End loop
```

En este ejemplo, el código de la izquierda genera 12 cuadrados igualmente espaciados. En el código de la derecha, las únicas diferencias son el borrar la pantalla (*clear*) justo antes de dibujar los cuadrados, y se añadir pausas de 20 milésimas de segundo (*delay*) al final de cada iteración. Los aspectos geométricos de ambas situaciones son esencialmente los mismos, y la ilusión de movimiento se genera al tener sólo un cuadrado visible. Esto deja en evidencia una conexión muy natural que es posible realizar entre dibujos y animaciones, lo que es un aspecto clave de la secuencia didáctica con la que se articuló el curso.

Categoría 2: Animación de imágenes



- Mover una imagen por un trayecto, mostrándola en distintas posiciones predefinidas.
- Mover una imagen gradualmente de una posición a otra.
- Mover varias imágenes gradual y simultáneamente.
- Mover una imagen gradualmente por una ruta predeterminada.

Si bien en la categoría anterior se consideran animaciones, estas cuentan con dos limitaciones fundamentales. En primer lugar, requieren de borrar todo lo dibujado en cada iteración, lo que dificulta de manera excesiva la posibilidad de realizar animaciones simultaneas. En segundo lugar, no se están animando objetos, en el sentido de moverlos, sino más bien se están generando colecciones de objetos de los cuales sólo uno quedará visible en cada instante.

Las animaciones a las que se refiere esta categoría, corresponden a movimientos de imágenes en pantalla, estas usualmente descargadas de Internet, lo cual añade mayores posibilidades creativas, y que en términos geométricos involucra la misma lógica de los problemas de la categoría anterior, es decir, de los dibujos en el plano cartesiano.

Una primera dificultad que se enfrenta en este tipo de problemas es la de la gradualidad del movimiento; lo que esencialmente requiere de realizar muchos movimientos pequeños con pausas también pequeñas.

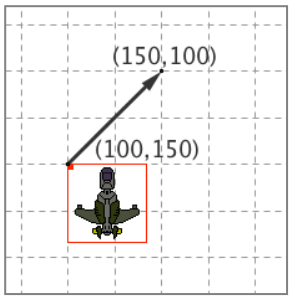
Ejemplo de movimiento con distinta gradualidad

```
Loop 200
        movespriteby("nave",1,2)
        delay(10)
End loop
```

```
Loop 10
        movespriteby("nave",20,40)  delay(200)
End loop
```

En este ejemplo, ambos códigos terminan trasladando la imagen "nave" según el mismo vector y en el mismo tiempo, sin embargo, el de la izquierda genera un movimiento mucho más fluido que el de la derecha. Los conceptos que surgen de esta situación son los de continuidad, gradualidad o fluidez; y dependen del vector de traslación, la cantidad de iteraciones y pausas intermedias. Para facilitar el hablar de esto con los estudiantes, se optó usualmente por utilizar la metáfora de los pasos, preguntándoles: "*Si la nave avanza 200 pixeles hacia la derecha, en diez pasos, ¿cuánto avanza en cada paso?*".

Lo anterior será una idea sumamente útil para controlar la animación de una imagen por una ruta predeterminada, como se propone en el siguiente ejercicio:

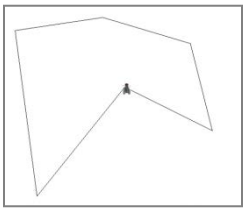


Una nave parte en la coordenada (100,150). En el siguiente código, completa los dos casilleros con los valores que permiten termine en la coordenada (150,100)

```
Loop 25
        moveSpriteBy(nave,2, -2)
        Delay(10)
End loop
```

Se han marcado en rojo las respuestas a esta pregunta, que consisten en las componentes del vector de traslación buscado. Para realizar este cálculo se debe considerar que en 25 pasos (*loop 25*) se debe avanzar 50 pixeles horizontalmente y retroceder 50 pixeles verticalmente.

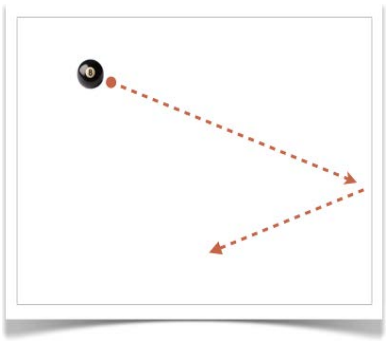
Con esta misma lógica se pueden calcular las distintas traslaciones necesarias para que una imagen siga una ruta determinada, como el siguiente ejemplo que es un desafío propuesto en todas las versiones del curso:



Dibuja el polígono cuyos vértices son los puntos A=(100,450); B=(300,200); C=(500,300); D=(450,100); E=(250,40) y F=(50,70). Luego, anima la mosca para que se recorra fluidamente el polígono, partiendo y terminando en A.

La estrategia más eficiente que utilizamos para resolver este tipo de problemas, consistió en identificar por cada tramo, cuánto se debe avanzar horizontal y verticalmente, y luego elegir una cantidad de pasos para ambos avances. Para esto último suele ser una buena elección el máximo común divisor, que permitirá realizar los avances en la mayor posible cantidad de pasos, logrando una mayor gradualidad de movimiento.

Categoría 3: Animación de rebotes



- Mover una imagen como si rebotara con un borde de la pantalla.
- Mover una imagen como si rebotara x veces con los bordes de la pantalla.
- Mantener una imagen rebotando indefinidamente con los bordes de la pantalla.
- Mover dos imágenes como si chocaran entre si y rebotaran.

Un tipo particular de animación, es la que se puede describir en la metáfora de distintos juegos como “rebote”. Si bien podría considerarse el rebote como parte de la categoría anterior, hay dos elementos esenciales para distinguirlos los de otras animaciones. El primero, es que se está simulando un movimiento de un objeto real y su interacción con otro, de manera que en este tipo de problemas serán una conexión apropiada para la idea de interacción. El segundo elemento, es que las distintas traslaciones involucradas están relacionadas entre sí, partiendo de la idea de que los ángulos de incidencia y reflexión son congruentes.

En la animación de rebotes, los elementos más naturales de interacción son los bordes de la pantalla. Si una imagen está avanzando según un vector determinado, simular estos rebotes involucra identificar cuánto debe avanzar para llegar a un borde de la pantalla y en qué dirección debe continuar posteriormente.

Ejemplo de rebote con el borde derecho de la pantalla

```
Loop 50
        movespriteby(“nave”,1,2)
        delay(10)
End loop
Loop 50
        movespriteby(“nave”,-
1,2)
        delay(10)
End loop
```

En este ejemplo, si se considera que la imagen “nave” se encuentra a 50 pixeles de distancia del

borde derecho de la pantalla, justo cuando termine la primera animación (el primer *Loop*), cambiará su trayectoria sólo en el sentido de la componente horizontal. Entonces, se estará simulando el rebote con el borde derecho de la pantalla. Si se encontrara a mayor distancia, entonces la cantidad de iteraciones deberá aumentar, y además deberá tomarse en consideración el ancho de la imagen.

Los rebotes con los otros tres bordes de la pantalla son análogos al anterior, debiendo considerarse que cuando la imagen rebota por la izquierda o derecha cambiará su componente horizontal, mientras que al rebotar con los bordes superior e inferior, deberá cambiar la componente vertical.

Con esta lógica se propone desarrollar un desafío como el siguiente: "*Realiza la animación de una pelota que choca 8 veces con los bordes de la pantalla (partiendo desde el centro).*" Ahora bien, la generalización de este problema consistiría en mantener la pelota rebotando indefinidamente, por ejemplo:

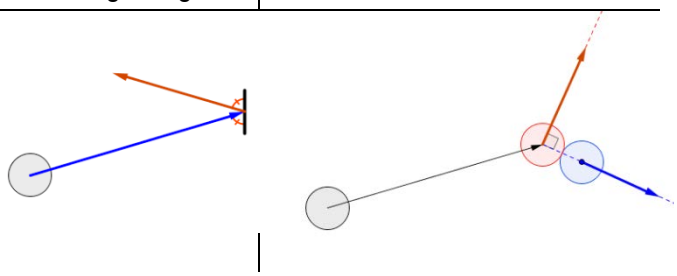
Ejemplo de rebote "infinito" con los bordes de la pantalla

```
MoveSpriteBy("nave",x,y)
If GetSpriteLeft("nave") > 1024 Then
        x= -x
End If
If GetSpriteLeft("nave") < 0 Then
        x= -x
End If
If GetSpritetop("nave") > 768 Then
        y=-y
End If
If GetSpritetop("nave") < 0 Then
        y=-y
End If
```

En este código es importante notar que la descripción de la traslación es algebraica, en vez de numérica, puesto que se está describiendo una solución general al problema, es decir, referida a cualquier traslación en una pantalla de dimensiones 1024x768. Además, las instrucciones *getSpriteLeft* y *getSpriteTop*, entregan las distancias a los bordes izquierdo y superior de la pantalla, luego, dependiendo de tales valores el vector de traslación debe cambiar.

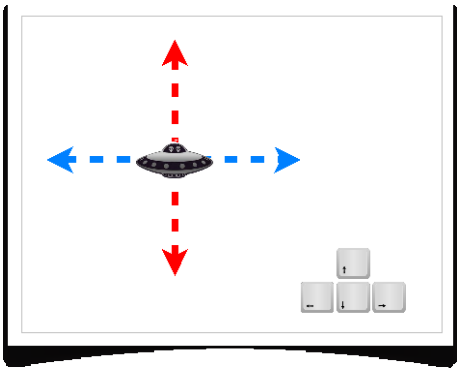
En términos didácticos, esencialmente se debe identificar en qué evento o dadas qué condiciones debe cambiar el vector de traslación y de qué manera, lo que luego se buscará describir en el lenguaje de programación que se desprende del lenguaje algebraico.

Pero existen otros elementos con los que un móvil puede interactuar, lo que dependerá de la metáfora del juego. Por ejemplo, otras "naves", "asteroides", "misiles", etc. Si dos móviles colisionan, o un móvil choca con un objeto estático, el comportamiento "ideal" de estos objetos dependerá de la o las trayectorias previas. Los problemas asociados a esta situación son diversos, pero en gran medida dependen de las características de los objetos involucrados.

Ejemplo de choque en Ping Pong	Ejemplo de choque en el Pool
	

En el ejemplo de la izquierda, un juego de Ping Pong, una “pelota” choca con una “paleta”, de manera que el tipo de rebote que se busca simular es análogo al de los bordes de la pantalla. En cambio, el ejemplo de la derecha, un juego de Pool, el choque de una bola con otra estática, genera el desplazamiento de ambas en trayectorias que están relacionadas con las rectas tangente y normal en el punto de contacto.

Categoría 4: Controlar imágenes con el teclado



- Mover una imagen con el teclado en cuatro direcciones posibles.
- Mover dos o más imágenes con el teclado en cuatro direcciones posibles.
- Mover una imagen con el teclado, sólo si se mantiene dentro de cierta región.
- Mover orientadamente una imagen con el teclado en x direcciones predefinidas.
- Mover una imagen controlando el giro gradual (avanza con □ y gira con □ y □)

La base de la mayoría de los juegos desarrollados está en que el usuario controle con el teclado el movimiento de uno o varios objetos. Con tal objetivo es que, usualmente cerca de la mitad del curso, se trabajaba en mover una imagen en cuatro direcciones posibles con el teclado, lo que denominábamos el “prototipo” del primer juego.

Controlar una imagen con el teclado requiere esencialmente de describir cómo deben cambiar su posición dependiendo de la tecla que se presiona. Un esquema de mover una imagen en cuatro direcciones, se puede reducir a:

Mover una imagen en cuatro direcciones

```
MoveSpriteBy("nave",x,y)
If iskeydown("up") Then
        y = y – 1
End If
If iskeydown("down")
Then    y = y + 1
End If
If iskeydown("left") Then
        x = x – 1
End If
If iskeydown("right") Then
        x = x – 1
End If
```

En este ejemplo la imagen “nave” se está moviendo a la coordenada *(x,y)*.

Los cuatro bloques de código siguientes determinan, por ejemplo, al presionar la tecla arriba (*up*), la ordenada disminuye; o bien, al presionar la tecla derecha (*right*) la abscisa se incrementa.

Una variante de este problema, consiste en definir una mayor cantidad de direcciones posibles, por ejemplo, 8 direcciones, lo que

involucra considerar la conjunción de dos de los casos del ejemplo anterior.

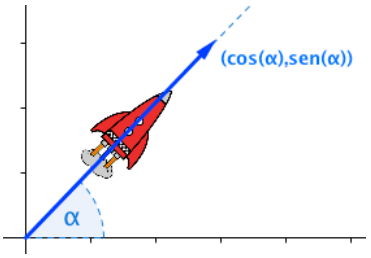
Lograr que dos imágenes se puedan controlar simultáneamente (*2 player*), involucra las mismas ideas antes expuestas, pero con más variables y eventos a considerar.

Además, si se busca que las imágenes avancen de manera orientada, es decir, que siempre apunten en la dirección que avanzan, se necesitará girarlas dependiendo de los eventos que corresponda. Sin embargo, esta última característica puede generar algunas dificultades si no se consideran también la posibilidad de dos o más eventos ocurriendo simultáneamente (dos o más teclas presionadas simultáneamente).

Por ejemplo si una determinada imagen debe estar rotada en 0º al presionar la tecla arriba, y en 90º al presionar la tecla derecha; al presionar ambas se genera un conflicto. En cuanto al vector de traslación, como se están incrementando ambas componentes del vector, la imagen avanzará en 45º, por lo que debiera orientarse en tal dirección. En consecuencia, esto requiere de considerar las distintas posibilidades de conjunción de eventos, y los ángulos que corresponden.

Un modelo de movimiento más complejo es el que involucra su avance y retroceso en cualquier dirección posible. Por ejemplo, como en muchos juegos, el lograr que avance con la tecla arriba, retroceda con la tecla abajo, y gire con las teclas izquierda y derecha. La complejidad de este tipo de movimiento está en identificar un vector de traslación asociado a un ángulo determinado, es decir, se requiere convertir las coordenadas de la "nave" a coordenadas polares.

Mover una imagen orientadamente

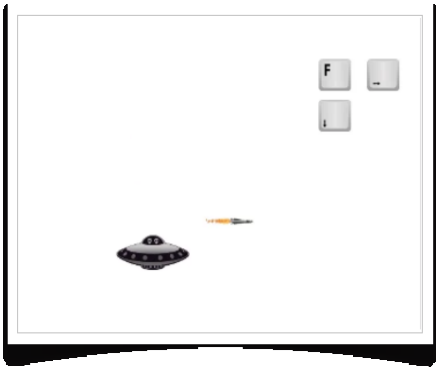


Si una imagen está orientada en αº, para que avance 1 en tal dirección, es necesario trasladarla según el vector:

$$\vec{v} = (\cos(\alpha), sen(\alpha))$$

Considerando que el entorno KPL no utiliza más que coordenadas rectangulares y ángulos en grados, se hace relevante, además de la transformación de coordenadas, la conversión de medidas de ángulos a radianes.

Categoría 5: Juegos que involucran disparos



- Disparar un misil desde una nave en la dirección actual.
- Mover una imagen por pantalla y reiniciarla al salir de pantalla
- Disparar un misil desde una nave en una de "x" direcciones predeterminadas
- Lograr que un misil siga a la nave.
- Lograr que un enemigo o misil siga a la nave.

Los "disparos" son una característica usual de muchos juegos, cuyo comportamiento tiene relación directa con cómo se mueven las "naves" desde donde se disparan. El diseño más usual con el que se partía, es de una imagen que se mueve

en cuatro direcciones, y dispara de izquierda a derecha.

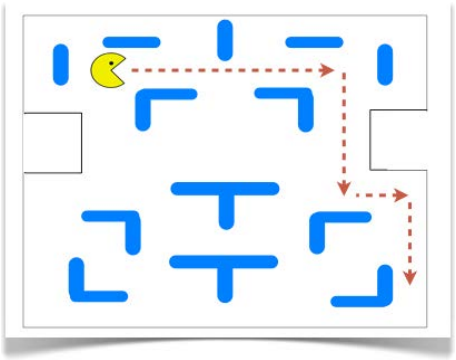
Simular un disparo requiere de varias consideraciones. Primero, la imagen que se dispara, debe partir de la posición actual de la nave; además debe avanzar hasta salir de la pantalla y ser reutilizable, para que sea posible disparar nuevamente no existan tantas imágenes como disparos se hayan hecho. Esto último involucra "reiniciar" la situación de la que el disparo partió, es decir, devolverlo a su posición inicial cuando salga de pantalla.

El nivel de complejidad de los juegos, cuando se incorpora este tipo de característica aumenta significativamente. Para ilustrar esto, considérese el siguiente diagrama de flujo.

En este problema se deben identificar diversas posibilidades, por ejemplo, cuando el disparo está avanzando, pero no está visible; o bien, cuando está en pantalla, pero no avanzando.
Cabe destacar que, "devolver" el disparo al principio, es ubicarlo en la posición de la que debe partir, es decir, en la misma posición que la nave, aunque oculto.

Una variante más compleja es la de un disparo de una imagen que se mueve de manera orientada, de manera que existirían más de una posible forma de que el disparo avance. En tal caso, dependerá de la orientación actual de la nave, lo que es una variable más, o sea, una nueva pregunta más, en el esquema anterior.

Categoría 6: Animaciones con condiciones



- Lograr que una imagen sólo avance por un camino dado.
- Evitar que una imagen avance sobre otra
- Simular un campo gravitatorio, en torno a una imagen
- Hacer que un personaje salte al presionar una tecla
- Hacer que un personaje suba por una rampla.

En esta última categoría se agrupan problemas más diversos, pero lo que tienen en común es que involucran definir condiciones en las que, ya sea, se realiza una cierta animación, o bien, se impide.

Ya se ha discutido previamente el problema de mover una imagen por una ruta determinada, pero algo más complejo es lograr que una imagen sólo pueda moverse por tal ruta. Esto se aplica a diversos juegos que involucran laberintos, y

requieren de determinar cuándo una imagen está en el camino o fuera de él.

Una opción es contar con imágenes que conforman el laberinto, y al detectar alguna intersección con la imagen móvil, deshacer el último movimiento. Esta idea también se puede aplicar a ubicar obstáculos en alguna ruta, o limitar una imagen a una región no rectangular de la pantalla.

Otra situación que involucra un problema similar, es lograr que una imagen se desplace sobre otra, por ejemplo, el lograr que un personaje avance sobre una determinada superficie. En tal caso, al no detectarse la intersección de ambas imágenes, deberá activarse una acción, como la caída de la imagen.

También surgen problemas asociados a movimientos que tienen una cierta duración, y por lo tanto requieren de definir condiciones de tiempo o bien determinar cuando han finalizado. Un caso clásico es el salto de Mario (*Mario Bros*.) que combinado con el avance del personaje tiende a describir una parábola.

Así, puede considerarse que al presionar una determinada tecla, se mueva describiendo la parábola, o bien, que sólo su altura se comporte de tal manera, completándose la parábola al combinarlo con el botón de avance.

6 RELACIÓN CON EL CURRÍCULUM MATEMÁTICO

Como se detalló en la sección anterior, los principales problemas, identificados a partir del análisis de los códigos fuente de los programas desarrollados por los estudiantes, se agrupan en 6 categorías. Cada una de éstas, agrupa problemas que surgen de los diseños de videojuegos elegidos por los estudiantes, ya sea, porque son parte de la estructura de tales juegos, o bien, porque surgen de la forma particular que ellos decidieron implementarlos. Correspondientemente a estos dos tipos de problemas, se podrían caracterizar a su vez como problemas preseleccionados y problemas emergentes.

Desde un punto de vista matemático, la literatura establece que los principales contenidos involucrados son de Geometría y Álgebra, estando lo geométrico vinculado principalmente a describir o simular movimientos (Kodicek, 2005). Lo algebraico, en cambio, puede responder a condiciones subyacentes a los juegos que no son necesariamente visibles.

6.1 RELACIÓN CON CONTENIDOS MATEMÁTICOS

De los problemas identificados, los temas matemáticos que se desprenden son:

Categoría	Temas matemáticos asociados
Dibujos en el plano cartesiano	• Coordenadas en el plano cartesiano. • Secuencias numéricas.
Animación de imágenes	• Coordenadas en el plano cartesiano. • Transformaciones isométricas. • Pendiente entre dos puntos. • Vectores.
Animación de rebotes	• Transformaciones isométricas • Vectores y reflexiones respecto a paralelas a los ejes coordenados. • Recta tangente y recta normal; y vectores asociados.
Controlar imágenes con el teclado	• Lenguaje algebraico • Lógica (conjunción y disyunción) • Coordenadas polares
Juegos que involucran disparos y Animaciones con condiciones	• Lenguaje algebraico. • Vectores. • Lógica

6.2 RELACIÓN CON LOS MAPAS DE PROGRESO DE APRENDIZAJE Y PISA

Atendiendo a los estándares de educación matemática del currículum chileno, los Mapas de Progreso de Aprendizaje (MINEDUC, 2010), es posible relacionar las categorías con los estándares explicitados sólo para Geometría y Álgebra, a saber:

Categoría	Álgebra	Geometría
Dibujos en el plano cartesiano		Nivel 4 Comprende el concepto de transformación isométrica y aplica estas transformaciones a figuras planas.
Animación de imágenes	Nivel 2 Identifica, describe y continua patrones numéricos y geométricos con figuras conocidas, mencionando alguna regla que genere la secuencia.	Nivel 4 Comprende el concepto de transformación isométrica y aplica estas transformaciones a figuras planas. Nivel 5 Comprende el concepto de transformación en el plano cartesiano, y utiliza la representación vectorial para describir traslaciones y homotecias de figuras geométricas en el plano. Formula y verifica conjeturas en relación a los efectos de la aplicación de una transformación a una figura en el plano cartesiano.
Animación de rebotes	Nivel 3 Comprende que en las expresiones algebraicas las letras pueden representar distintos valores de acuerdo al contexto. Nivel 4 Traduce expresiones desde el lenguaje natural al lenguaje matemático y viceversa.	
Controlar imágenes con el teclado		
Juegos que involucran disparos y Animaciones con condiciones	Nivel 7 Modela situaciones o fenómenos provenientes de diversos contextos.	Nivel 7 Resuelve problemas geométricos estableciendo relaciones entre conceptos, técnicas y procedimientos de distintas áreas de la matemática.

Dado que los estándares chilenos son integrados es importante distinguir que algunas de estas menciones son relativas a procesos, como en los niveles 7 de álgebra y geometría, mientras que en los otros niveles mencionados la relación es en cuanto a contenido.

En cambio considerando las competencias seleccionadas para la prueba PISA (Rico, 2006), los todos problemas identificados conservan una relación más bien transversal, con la mayoría, a saber:

- La competencia de **comunicar** se observa en todos los problemas, a partir de la necesidad de, por un lado, organizar el trabajo con los pares, como también el elaborar correctamente las situaciones problemáticas para pedir ayuda a los profesores. Especialmente en cuanto a los problemas emergentes, la necesidad elaborar las situaciones, que usualmente se observan de manera gráfica, cumple un rol fundamental en la formulación de los problemas.
- La competencia de **modelar** está presente también transversalmente, especialmente en la segunda etapa del modelo de Polya, cuando los estudiantes deben elaborar un plan para resolver el problema que estén enfrentando. En gran medida, como se ha visto previamente, los problemas de los videojuegos son desafíos gráficos que involucran simular movimientos físicos, de manera que el modelamiento aquí se entienden en el sentido de traducir el fenómeno físico a modelar, al lenguaje matemático y de programación.
- El **uso de lenguaje simbólico, formal y técnico**, tiene una variante especial en el contexto de la programación, dado que el lenguaje que se utiliza (como cualquier otro) requiere de ciertas normas de sintaxis relativamente estrictas. De esta forma, según la tercera etapa del modelo de Polya, la forma como los estudiantes ejecutan el plan, es escribiendo los códigos para que el programa los ejecute.
- Finalmente **plantear y resolver problemas** ocurre especialmente con los problemas emergentes, aunque también está presente en los problemas preseleccionados. En gran medida el marco que determina la ocurrencia de los problemas, surge de las reglas de los juegos que eligen los estudiantes, por ejemplo, para lograr articular las condiciones para ganar o perder.

6.3 RELACIÓN CON LOS ESTÁNDARES DE LA NCTM

En el esquema doble de estándares del National Council of Teachers of Mathematics (NCTM, 2000), los problemas identificados se relacionan con los de contenido y de proceso. En cuanto a los contenidos, el panorama es análogo al de los MPA chilenos, observándose relaciones explicitas con los de Álgebra y Geometría:

Estándares de contenido	
Álgebra	• Comprender patrones, relaciones y funciones; • Representar y analizar situaciones y estructuras matemáticas usando símbolos algebraicos; • usar modelos matemáticos para representar y comprender relaciones cuantitativas;
Geometría	• especificar posiciones y describir relaciones espaciales usando geometría de coordenadas y otros sistemas de representación; • aplicar transformaciones y usar la simetría para analizar situaciones matemáticas; • usar la visualización, el razonamiento espacial, y la modelización geométrica para resolver problemas.

En cuanto a los estándares de proceso, se observa una situación similar a la que se da con las competencias PISA, es decir, con una relación más

bien transversal que se relaciona con todas las categorías de problemas identificados:

Estándares de procesos	
Comunicaciones	• comunicar su pensamiento matemático de manera coherente y clara a los compañeros, profesores y a otras personas; • analizar y evaluar el pensamiento matemático y las estrategias de los demás; • usar el lenguaje de las matemáticas para expresar ideas matemáticas de manera precisa.
Conexiones	• reconocer y aplicar las ideas matemáticas en contextos no matemáticos.
Representaciones	• usar representaciones para modelizar e interpretar fenómenos físicos, sociales y matemáticos
Resolución de Problemas	• resolver problemas que surgen de las matemáticas y en otros contextos; • aplicar y adaptar una variedad de estrategias apropiadas para resolver problemas;

7 CONCLUSIONES

Si bien la categorización de problemas tiene cierta relación con los contenidos seleccionados y su organización en unidades, gran parte de los problemas detectados son problemas emergentes, es decir, problemas que surgieron producto de aspectos no considerados en las guías de aprendizaje. Esto hizo que el curso fuera constantemente desafiante, para estudiantes y profesores, y nutrió permanentemente de ideas nuevas las clases, todo esto desde los intereses de los estudiantes, de sus juegos favoritos o de los que se les ocurriera inventar.

Esta flexibilidad de las actividades del curso, que se traduce en que los estudiantes contaran con plena libertad para programar los juegos que quisieran implicó elevar el nivel de dificultad permanentemente, siendo que usualmente quien es garante de la dificultad es el profesor, en esta experiencia los son estudiantes, característica frecuentemente asociada a los talentos académicos.

La dificultad de los problemas emergentes está también asociada a aspectos particulares de la programación y su enseñanza. Cuando se cuenta con un videojuego parcialmente programado, y se busca incorporar una nueva característica, es común que esto obligue a adaptar todo lo que se ha creado. En nuestro caso, la incorporación del "disparo de un misil" a un prototipo muchas veces requirió de adaptar variables y sentencias preexistentes para que tal característica funcionara correctamente. Así, la creencia de los estudiantes de que tal característica simplemente se debía "agregar", se veía frecuentemente desafiada por la experiencia de incorporar elementos nuevos que interactuaban con los anteriores.

El aspecto más interesante que se dio en este curso, y cuya base teórica está asociada al modelo de Polya y los aportes de Borasi, es la forma en la que aparecen muchos problemas

emergentes, esto es, gráficamente. Al incorporar una determinada característica, como se describía previamente, o al ejecutar un plan, frecuentemente los estudiantes se encontraron con que gráficamente lo que veían no correspondía con lo que esperaban que sucediera. Luego, tales problemas surgen sin una formulación, y ante tal dificultad se les genera la necesidad de formularlos para pedirle ayuda a los profesores o a sus compañeros. En esta etapa, con el tiempo fuimos descubriendo que la forma de fomentar la formulación del problema, que sería la meta de la primera etapa según el modelo de Polya, era dialogando con ellos. Esto es algo un tanto inusual en relación a la típica experiencia escolar en la que la mayoría de los problemas que se les presentan, están formulados y de manera muy correcta y secuenciada. En tal contexto entender el problema es en gran medida una tarea de lectura comprensiva, mientras que en el contexto de la programación de juegos entender el problema es algo mucho más desafiante.

En cuanto a la relación con los estándares de educación matemática, el análisis curricular estuvo dado por identificar con qué contenidos y qué procesos se relaciona y así se observa una clara relación con la geometría analítica y transformacional, presente en el currículum chileno, una relación menor con el álgebra en términos del contenido típicamente descrito como lenguaje algebraico (o traducción de) y secuencias numéricas, y una relación ocasional con la lógica proposicional.

De manera similar, se da la relación con los estándares de contenido del NCTM, aunque en estos la presencia de las coordenadas cartesianas es más explícita, al indicar *"especificar posiciones y describir relaciones espaciales usando geometría de coordenadas y otros sistemas de representación".*

En cuanto a competencias, para efectos de la prueba PISA y estándares de proceso del NCTM, la relación que se da con los problemas identificados es transversal, destacando la competencia de comunicar, modelar, usar lenguaje formal y plantear y resolver problemas. En muchas ocasiones estas competencias se presentan (o necesitan) prácticamente en esta misma secuencia, como parte de las etapas del modelo de Polya, es decir: requieren comunicar un problema emergente para terminar formulándolo y elaborar un plan; y de usar lenguaje formal para ejecutar el plan.

Es claro, entonces, que existen una relación sistemática en cuanto a los estándares de proceso, más que los de contenido, y por tal razón es quizás poco práctico implementar este tipo de iniciativas para enseñar matemática de manera formal, esto es, poniendo la programación de juegos al servicio del currículum. Sin embargo, en ningún caso esto significa que aprender a programar juegos sea poco relevante, sino que propuesta de manera utilitaria para el currículum de matemática, tiene a perder potencia. En gran medida esta reflexión la realizo inspirado por el artículo de Seymour Papert "What's de big idea?", en español, ¿Cuál es la gran idea?:

> "La clave para comprender por qué la escuela es lo que es, reside en reconocer una tendencia sistemática a deformar las ideas deformas específicas, para ajustarlas en un marco pedagógico. Una de estas deformaciones la describo como desempoderar ideas. Este artículo propone una nueva dirección para la innovación en la educación: el re-empoderamiento de las ideas menos favorecidas"
>
> (Papert, 2000)

Mi interpretación del proyecto de re-empoderamiento de las ideas matemáticas, después de cinco años de enseñarle a programar videojuegos a niños con talento académico es esta: el verdadero objetivo no es enseñar matemática, sino enseñar a pensar, y esto requiere de pensar en este tipo de iniciativas no al servicio de la matemática escolar, sino al servicio de lo que la matemática escolar finalmente persigue. He ahí el principal potencial de todos estos esfuerzos.

BIBLIOGRAFÍA

Alfaro, C., & Barrantes, H. (2008). ¿Qué es un problema matemático Cuadernos de investigación y formación en educación matemática.

Barrionuevo, A. (2009). "Lenguajes de programación para niños". Revista digital enfoques educativos No 51, 2009: 29-39.

Begel, A., & Klopfer, E. (2004). StarLOgo TNG: An introduction to game developments, 1–15.

Borasi, R. (1986). On the nature of problems. Educational studies in Mathematics.

Kodicek (2005). Mathematics and Physics for Programmers. Charles River

MINEDUC. (2009). "Marco curricular - Sector matemática".

MINEDUC. (2010). "Mapas de progreso del aprendizaje - Sector matemática".

National Council of Teachers of Mathematics NCTM, (2000). Principles and Standards for School Mathematics. Reston, Va.

Papert, S. (2000). What's the big idea? Towards a pedagogy of idea power. IBM Systems Journal, 39 (3/4). Disponible en: http://llk.media.mit.edu/courses/readings/Papert-Big-Idea.pdf

Polya, G. (1985). Cómo plantear y resolver problemas. Méjico: Trillas.

Rico, L. (2006). Marco teórico de evaluación en PISA sobre matemáticas y resolución de problemas. Revista de Educación, 275-294. Ministerio de Educación y Ciencia. España.

Schwartz, J. (2005). Empezando a programar con KPL, 1–26. Retrieved from http://www.kidsprogramminglanguage.com

Schwartz, J. (2006). Kid's Programming Language (KPL), 1–4.

Schwartz, J. (2009). Phroglab: Talking a code-centric approach to introductory CS education, 1–5.