Lessons learned: use of interactive graphics to support contextualized programming education

Armando Arce-Orozco*†
*School of Computing
Costa Rica Institute of Technology
Cartago, Costa Rica
arce@tec.ac.cr
†School of Informatics
National University
Heredia, Costa Rica
ararce@una.ac.cr

Antonio González-Torres‡§
‡ Dept. of Computer Engineering
Costa Rica Institute of Technology
Cartago, Costa Rica
antonio.gonzalez@tec.ac.cr
§ Faculty of Engineering
ULACIT
San José, Costa Rica
agonzalez@ulacit.ac.cr

***Abstract*—Learn to program is a difficult task that requires to be highly motivated to get confronted to abstract and complex problems and thus several teaching strategies have been proposed to assist students on how to grasp the intricacies of methods, structures and algorithms involved in programming. One of these strategies is teaching programming in a contextualized manner through the use of interactive graphics to provide visual aids for the easy understanding and learning of some basic programming concepts. The use of interactive graphics provides immediate feedback to students on the concepts they are learning, in an entertained and contextualized way, through their active participation through the creation of images and figures that they can manipulate. This paper discusses the experiences of the authors in the use of strategies to facilitate the learning of programming structures in a contextualized learning environment, supported by the use of interactive graphics as a means to increase their comprehension and the interest of students. The discussion includes the use of SVG graphics and two environments developed by the authors to provide contextualized learning.**

***Index Terms*—Interactive graphics, contextualized learning, graphics environments, programming education**

I. Introducción

La adquisición de conocimientos de programación es una de las principales competencias que los estudiantes universitarios de carreras afines a la computación deben adquirir durante el transcurso de sus estudios. El proceso de enseñanza y aprendizaje de las habilidades para programar no es trivial, requiere compromiso y dedicación para modelar la forma de razonamiento y desarrollar las competencias lógicas de los estudiantes, e implica que tanto estos como los profesores deben formar un equipo de trabajo. Por un lado los profesores deben propiciar un ambiente adecuado y plantear las actividades de formación apropiadas y, por otro lado los alumnos deben disponer del tiempo y concentración necesaria para llevarlas a cabo. Esto supone que los profesores deben conocer los aspectos técnicos, pero también deben tener claridad metodológica para enseñar a pensar y razonar de forma independiente a cualquier lenguaje de programación.

Las competencias que los estudiantes deben desarrollar se pueden clasificar como competencias de alto y bajo nivel. Las competencias de alto nivel involucran la capacidad de abstracción [2], planificación, descomposición de problemas, razonamiento condicional y analógico, y la depuración de programas [13]. En tanto, entre las competencias de bajo nivel se encuentran el uso de estructuras condicionales y de repetición [14] [6], vectores, matrices, listas, pilas, colas, grafos, árboles, recursividad, y la comprensión de conceptos relacionados con la herencia, implementación de interfaces, encapsulación y polimorfismo [7].

La enseñanza de estos temas es un reto para las universidades, sobretodo si se tiene en consideración que un gran número de docentes utiliza métodos de enseñanza descontextualizada. Este tipo de estrategias no ofrecen detalles sobre su aplicación a situaciones reales o específicas, no contemplan la interacción activa entre los profesores y los alumnos, y su utilidad se circunscribe a la transmisión de conocimientos generales. Esto influye en que algunos estudiantes tengan dificultades durante el aprendizaje de conceptos elementales de programación [21], lo cual puede deberse a su falta de interés [1] [4] por considerar que programar es tedioso, aburrido y asociado a un gran número de conceptos abstractos sin aplicación concreta [2]. Lo anterior afecta la disposición de estos estudiantes para invertir tiempo adicional en comprender los conceptos por no entender cómo éstos serán de utilidad en los cursos de niveles superiores y en sus futuros trabajos.

Como resultado, se han propuesto múltiples enfoques para realizar la contextualización en la enseñanza de la programación. En general, estos enfoques buscan relacionar los principios y conceptos son su aplicación a entornos reales, que sean familiares o de fácil comprensión por parte de los estudiantes. El fin que persiguen estas estrategias es motivar a los alumnos para que aprendan, a la vez que comprenden las razones por las cuales es importante lo que están estudiando, y en qué circunstancias puede ser de su utilidad [3].

El objetivo de este trabajo es describir y discutir las experiencias de los autores en el planteamiento de estrategias y

herramientas para facilitar el aprendizaje y la comprensión de estructuras, métodos y técnicas de programación en un entorno de aprendizaje contextualizado mediante el uso de gráficos interactivos como un medio para incrementar el interés y comprensión de los estudiantes. La razón por la cual se utilizaron gráficos interactivos como medio para la contextualización, es que la programación de estos requiere el uso de diversos algoritmos y estructuras para representar datos (concretos o abstractos). Además, el uso de representaciones visuales puede proporcionar a los alumnos información, al permitirles explorar y obtener conocimiento relevante de los datos con el uso de su propia creación, lo cual, por lo tanto, puede generar mayor interés y atención en los contenidos de las asignaturas, así como un mayor nivel de aprendizaje.

II. Antecedentes

La enseñanza de la programación de sistemas de software de forma contextualizada busca involucrar al estudiante de manera personal [3], con el fin de que sea percibida como una actividad que contribuye a mejorar el mundo. Cooper y Cunningham argumentan que diseñar cursos dentro de un contexto puede mejorar el aprendizaje, incrementar la motivación, facilitar el desarrollo de habilidades y la comprensión de conceptos complejos, así como incrementar el número de estudiantes que desean continuar sus carreras universitarias.

El contexto, para este tipo de aprendizaje, es la fuente de ejemplos, motivaciones e ideas para resolver problemas y realizar proyectos a lo largo de cualquier curso. Sin embargo, la enseñanza contextualizada requiere invertir un gran número de horas para elegir y describir un contexto adecuado, diseñar las lecciones y explicar a los alumnos el problema que se desea resolver y su relación con la realidad. Esto implica la preparación y desarrollo de más actividades para favorecer la comprensión de los temas, lo cual conlleva que los profesores requieren de tiempo adicional para estas tareas. Pero además, la enseñanza de los contenidos también demanda una mayor cantidad de horas, y puede tener como consecuencia que se excluyan algunos contenidos de relevancia, aunque es posible profundizar más en los temas que se estudian [9].

Tomando como base lo anterior, es necesario contar con el apoyo de estrategias y herramientas para facilitar la enseñanza contextualizada. Por esa razón se han propuesto diversas opciones para la enseñanza de la programación de forma contextualizada, entre las cuales se encuentran los gráficos interactivos. En este campo se ubican los esfuerzos efectuados por Seymour Papert [22], Erick Roberts [28] [27], John Maeda [19], Casey Reas y Ben Fry [26].

Seymour Papert [22] es uno de los creadores del Logo, un lenguaje de programación orientado a facilitar los procesos de enseñanza y aprendizaje de las matemáticas y la programación. La difusión de este lenguaje permitió que los gráficos interactivos fueran reconocidos como una herramienta útil para aprender con mayor facilidad algunos conceptos básicos de programación.

El uso de Logo, y otros lenguajes similares, permite que los estudiantes aprendan de forma contextualizada y entretenida creando imágenes y figuras que puede manipular de forma interactiva, a la vez que reciben retroalimentación inmediata sobre los conceptos que están aprendiendo. Este tipo de aprendizaje es activo y el estudiante es el principal actor de su formación.

Por su parte, el esfuerzo de Erick Roberts abarca el desarrollo de múltiples herramientas pedagógicas para cursos de nivel inicial e intermedio [28] [27], las cuales se componen de bibliotecas para la creación de gráficos simples. En estas bibliotecas se definen nueve tipos de objetos para representar figuras primitivas, además de un tipo de objeto especial para construir figuras más sofisticadas mediante la composición de figuras simples. Los objetos primitivos son arcos, imágenes, etiquetas, líneas, elipses, polígonos, rectángulos 2D, rectángulos 3D y rectángulos redondeados; y a todos ellos se les puede definir una serie de propiedades visuales

El objetivo de estas bibliotecas es que los estudiantes puedan reconocer la importancia del uso de parámetros cuando se invoca un métodos, como cuando se crea cualquier figura gráfica simple, como una línea o rectángulo, o se cambia el tamaño de un círculo. Además, los estudiantes pueden aprender sobre la modularidad y composición al ensamblar figuras compuestas a partir de las primitivas y al crear la estructura de un programa para desplegar los gráficos [28] [27]. En línea con esto, algunos investigadores [25] sugieren que el uso de herramientas gráficas debería ser obligatorio en los cursos introductorios de programación porque permiten que los estudiantes observen los resultados de la codificación de los ejercicios que están realizando.

En tanto, en una dirección similar John Maeda llevó a cabo un experimento pedagógico que buscaba facilitar el aprendizaje de la programación por parte de diseñadores, artistas y aquellos que sin estar cursando una carrera en computación tuvieran interés en programar [19]. El proyecto en el cual se enmarcó el experimentó se llamó “Diseño mediante Números” y contó con la participación de Casey Reas y Ben Fry, quienes se basan en las ideas de Maeda para desarrollar Processing [26], un entorno y lenguaje de programación para aprender a programar en el área de las artes visuales que cuenta con un gran número de usuarios que supera las decenas de miles.

Processing es procedimental y permite obtener resultados visuales de forma rápida sin tener que manipular un modelo de objetos. Este lenguaje utiliza el concepto de estado actual, por lo que al establecer propiedades, como colores o transformaciones geométricas, se aplican a todas las figuras que se dibujen posteriormente al momento en que fueron definidas. Así, a diferencia de las bibliotecas de gráficos desarrolladas por Roberts, no es necesario especificar todas las propiedades para cada figura.

El éxito de Processing ha servido de estímulo para la aparición de múltiples variantes e implementaciones como Processing.js [24], OpenFrameworks [23], Cinder [18] y P5.js [20]. En el caso de Processing.js y P5.js permiten generar gráficos interactivos en la web, pero la principal diferencia es que Processing.js usa una sintaxis parecida a la de Processing e incluso permite ejecutar la gran mayoría de los programas

escritos en este, en tanto que P5.js utiliza la sintaxis de Javascript. Por su parte, OpenFrameworks y Cinder permiten utilizar un conjunto de rutinas de dibujo similares a las de Processing, pero usando C++.

De forma adicional a los esfuerzos expuestos, existe un gran número de herramientas que los profesores puede utilizar para apoyar las estrategias de contextualización de la enseñanza de algoritmos, métodos y técnicas de programación, las cuales se enmarcan en ocho tipos diferentes de soluciones [17] que se presentan a continuación:

- **Micromundos (microworlds):** Al estudiante se le presenta un mundo simplificado en el que debe interactuar mediante un personaje que realiza acciones [10].
- **Contar historias (storytelling):** Permiten que los estudiantes implementen programas que representan animaciones para contar historias [11].
- **Computación de medios (media computation):** Se usan para la creación y procesamiento de multimedios mediante programación, como imágenes, sonidos, vídeos y páginas web. [8]
- **Juegos:** Algunas herramientas para la programación de juegos se basan específicamente en los conceptos de programación y pensamiento computacional, mientras que algunos juegos (por ejemplo, Minecraft) se pueden modificar y ampliar mediante programación. [15]
- **Robots:** Existen soluciones de software que permiten a los estudiantes controlar robots virtuales en la pantalla, en lugar de interactuar con un robot físico. [16]
- **Simulaciones:** Se pueden usar para crear simulaciones similares a los juegos, posiblemente con las mismas herramientas con las cuales estos se desarrollan. [15]
- **Inteligencia Artificial (IA):** Existen varios ambientes que se asemejan a los juegos, pero requieren que se escriba el código de IA de los personajes. [12]
- **Interfaces gráficas de usuario (GUI):** Estas herramientas permiten que el estudiante programe aplicaciones con interfaces de usuario sofisticadas. [27]

Es importante considerar que de forma independiente al tipo de herramienta que se utilice para la contextualización del aprendizaje, todas involucran, en mayor o menor grado, el uso de gráficos interactivos, lo cual incluye tanto imágenes como animaciones para hacer el aprendizaje de los estudiantes más atractivo.

III. Aproximaciones de enseñanza

El aprendizaje de la programación es una tarea que resulta compleja y ardua para muchos estudiantes universitarios. La dificultad del proceso de aprendizaje se deriva de la modificación que deben realizar en su forma de razonar para crear algoritmos con los detalles y pasos necesarios en el orden adecuado. A lo cual se debe agregar que deben adquirir las competencias para la solución de problemas abstractos en condiciones de ambigüedad. Esto es un reto todavía mayor para estudiantes que provienen de otras carreras que no son computación, y por lo tanto, los cursos de programación que reciben forman parte de su formación complementaria. Como resultado, en el 2016 se inició un esfuerzo para incorporar el uso de gráficos interactivos en los cursos optativos que la Escuela de Computación imparte, así como los que imparte para carreras de otras escuelas. En esta sección se describe el uso de SVG, VGDisplay y Diököl para la generación de gráficos 2D interactivos, como diferentes aproximaciones para apoyar la enseñanza de la programación en estos cursos.

A. *Gráficos mediante SVG*

La primera aproximación para incorporar la aplicación de gráficos interactivos se realizó en el 2016 durante el curso Análisis y Diseño de Algoritmos que se imparte a estudiantes de las carreras de Electrónica y Electromecánica con el uso de *SVG* (Scalable Vector Graphics). SVG es un estándar que permite especificar gráficos vectoriales que pueden ser mostrados por cualquier navegador Web y cuyo uso común consiste en incrustar la especificación de los gráficos en un archivo HTML, sin embargo, también es posible crear un archivo SVG de forma directa, a partir del detalle de la definición de un gráfico.

Los proyectos de programación que realizaban los estudiantes en dicho curso requerían crear diferentes tipos de gráficas estadísticas utilizando C++, aunque previamente sólo habían recibido un curso de Elementos de Computación utilizando Python. Los alumnos requerían leer archivos de datos en formato CSV, crear listas enlazadas, árboles y grafos a partir de los cuales debían generar las gráficas en formato *SVG* completamente en C++. Las gráficas estadísticas que se asignaron a los estudiantes incluían diagramas de barras, líneas, áreas, y pasteles. El procesamiento de la información para generar las gráficas involucraba tareas de ordenamiento, clasificación, y cálculo de resúmenes de datos. Las figuras 1 y 2 muestran ejemplos de algunos proyectos realizados por los estudiantes en este curso.

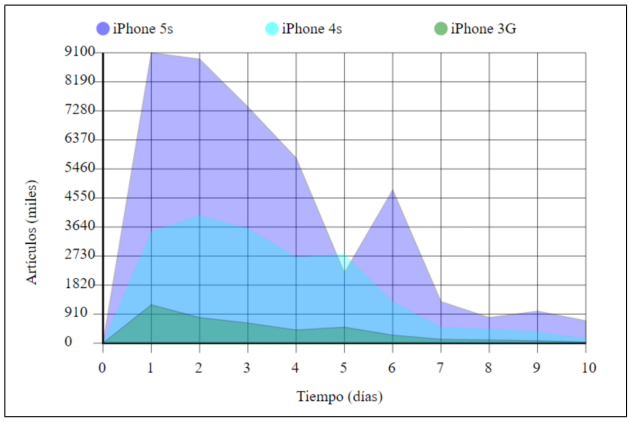

Fig. 1. Gráfica de áreas

Antes de asignar los proyectos programados se determinó que era necesario explicar a los estudiantes una serie de conceptos básicos relacionados con la generación de gráficos. Estos conceptos incluyen tipos de figuras de dibujo y sus parámetros (punto, línea, rectángulo, círculo, curva, polígono), propiedades visuales de las figuras (color de relleno, color

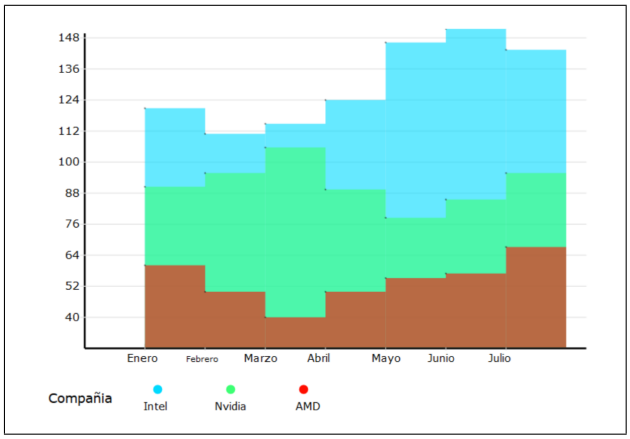

Fig. 2. Gráfica de áreas en escalera

de borde, ancho del borde, estilo de línea), transformaciones geométricas (traslación, escalamiento, rotación), composición de figuras más complejas (grupos), manejo de textos e interacción con las figuras.

El método que se utilizó para generar las gráficas en formato *SVG* consistió en escribir directamente la especificación de cada figura a la salida estándar de C++, o bien redireccionar dicha salida a un archivo de datos. Por ejemplo, el siguiente segmento de código muestra la forma de generar parte de la especificación:

```
int divx=100;
int divy=400;
int xdiv=xnum/10;
int ydiv=ynum/10;
for (int i=0;i<22;i++) {
  graph<<"<line x1=\"90\" y1=\""
   <<divy<<"\" x2=\"600\" y2=\""<<divy
   <<"\" style=\"stroke-width:0.5\"/>"
   <<"<text x=\"60\" y=\""<<divy+3<<"\">"
   <<yi<<"</text>"<<endl
  if(i!=11)
    yi+=ydiv;
  divy-=30;
}
```

Como se puede observar en este ejemplo, la definición de las figuras es complicada y propensa a errores. Otro problema adicional es que este método no permite realizar interacción con los datos, a menos que se incorporen bibliotecas adicionales en *Javascript*, pero esto se encontraba fuera del alcance y objetivos planteados. Estas desventajas afectaron en gran medida el interés de los estudiantes por este tipo de proyecto y no logró la motivación deseada en el curso porque de acuerdo con la opinión de algunos alumnos, el enfoque en lugar de simplificar el aprendizaje más bien lo complicó.

B. Biblioteca de gráficos - VGDisplay

A inicios del año 2017 se realizó un nuevo intento de incorporar el uso de gráficos interactivos en los cursos. Sin embargo, para evitar los problemas que se presentaron con anterioridad, se decidió desarrollar *VGDisplay* (Vector Graphics Display), una herramienta propia de dibujo de gráficos que se basa en el enfoque propuesto por Erick Roberts, pero utilizando el mecanismo de graficación de modo directo. En este modo de graficación cada vez que se especifica una instrucción el resultado se muestra de forma inmediata en la pantalla, a diferencia del modo retenido que primero crea el modelo de objetos y posteriormente lo utiliza para generar los diferentes gráficos en pantalla.

La herramienta acepta una serie de comandos sencillos y utiliza *OpenGL* para dibujar figuras en la pantalla. El conjunto de comandos que proporciona es una versión simplificada de los comandos de Processing. Cabe considerar que aunque *VGDisplay* fue desarrollado en C++, este es un programa independiente y una biblioteca que es usada por otra aplicación. La herramienta cuenta con versiones para los sistemas operativos MS Windows, Linux y MacOS, y su instalación requiere menos de 1 MB de espacio en disco.

El uso y evaluación de esta herramienta se realizó durante el curso de Diseño de Interfaces de Usuario que se impartió a inicios del 2017 y fue utilizada para elaborar bosquejos de diseños de pantallas que fueran interactivos. En este caso se le brindaba a los estudiantes una especificación de requerimientos de una aplicación sencilla y estos debían diseñar el conjunto de ventanas o pantallas de su solución al problema especificado. Un aspecto adicional que los alumnos debían contemplar es que las pantallas tenían que incorporar algún tipo de interacción simulada para mostrar las transiciones entre ventanas. Las figuras 3 y 4 muestran algunos ejemplos de los resultados obtenidos por algunos estudiantes de dicho curso.

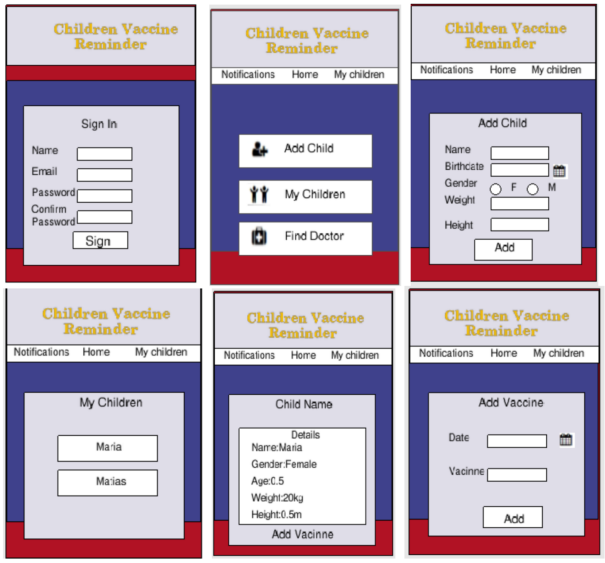

Fig. 3. Bosquejo de pantallas para control de vacunas

El ingreso de los comandos de dibujo en la herramienta se realiza usando dos métodos. El primer método consiste en proporcionar un archivo de comandos escrito con un editor de textos común. El contenido de un archivo de este tipo es similar al que se muestra a continuación:

```
fill 145 233 247 250
```

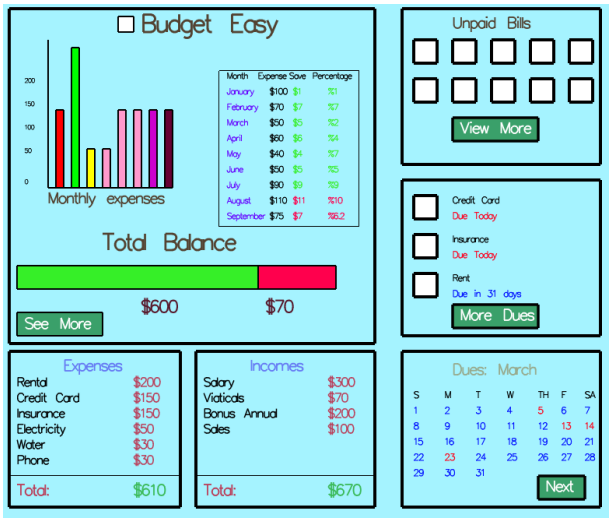

Fig. 4. Bosquejo de pantalla para tablero de mando

```
rect 320 10 100 30
rect 425 10 100 30
rect 530 10 100 30
fill 0 0 0 100
textSize 14
text "Obras" 353 30
text "Colecciones" 440 30
text "Perfil" 565 30
```

El segundo método, que resulta más interesante, consiste en el uso de comunicaciones mediante sockets, para lo cual se crearon, adicionalmente, dos pequeñas bibliotecas en C++ y Python, para comunicarse con el ambiente *VGDisplay*. Para usar este mecanismo el estudiante debe importar esta pequeña biblioteca en su aplicación y así poder invocar las capacidades de dibujo de la herramienta.

Una ventaja que se obtiene al utilizar el segundo método es la interacción, porque por medio del socket se pueden enviar de vuelta los eventos que genera el usuario, como presionar o mover el mouse, y teclear. El siguiente fragmento de código muestra la forma como se pueden generar los comandos gráficos desde Python (se puede notar la semejanza con la estructura de una aplicación en Processing):

```
import p5d
def setup():
  pg.size(480, 270)
def draw():
  pg.background(255)
  pg.ellipseMode(pg.CENTER)
  pg.rectMode(pg.CENTER)
  pg.stroke(0)
  pg.fill(150)
  pg.rect(240, 145, 20, 100)
pg = p5d.PGraphics()
pg.setupFunc(setup)
pg.drawFunc(draw)
pg.listen()
```

Aunque esta nueva herramienta facilitó a los estudiantes la creación de bosquejos de pantalla de forma rápida y sencilla, se presentaron varios problemas. El primer problema tiene relación con el uso de *VGDisplay* y el ambiente de programación (Python ó C++) de forma conjunta. Esto provoca que se generen errores si *VGDisplay* no se encuentra en ejecución antes que se inicie el programa que se desea probar. Otro problema es que las aplicaciones con una interacción intensa sufren retrasos debido a la comunicación mediante sockets. Esto no sucede normalmente con acciones sencillas como clics en la pantalla, pero algunos eventos, como arrastrar objetos o realizar animaciones ralentizan considerablemente la aplicación.

C. Ambiente de programación de gráficos - Diököl

A finales del 2017 se comenzó a trabajar en una nueva herramienta para la generación de gráficos interactivos en modo directo que incorpore un lenguaje de scripts. El principal requerimiento que se planteó para esta herramienta es que debe servir como un ambiente completo de generación de visualizaciones sofisticadas, sin requerir de un lenguaje externo. El nombre que se le otorgó al ambiente es *Diököl*, que significa ”figura” en un lenguaje aborigen de Costa Rica. En este ambiente se incorporó el uso de LuaJIT que consiste en un intérprete optimizado del lenguaje Lua y que permite escribir aplicaciones completas dentro del mismo ambiente.

A diferencia de VGDisplay, *Diököl* utiliza la biblioteca ShivaVG para generar gráficos vectoriales que consiste de una implementación en OpenGL del estándar OpenVG para gráficos en 2D. Una característica clave de este ambiente es que requiere de menos de 1.5 MB de almacenamiento y cuenta con versiones para ambientes MS Windows, Linux y MacOS, lo cual hace que una aplicación escrita en este pueda ejecutarse de forma transparente en cualquiera de estos sistemas operativos.

Diököl se utilizó por primera vez durante el curso de Visualización de Información, a inicios del 2018. Con el fin de familiarizar a los estudiantes con el lenguaje y comandos gráficos se elaboraron una serie de laboratorios que consistían en realizar una adaptación de pinturas abstractas de artistas famosos mediante animaciones. Las figuras 5 y 6 muestran dos ejemplos de las adaptaciones realizadas por los estudiantes en este curso.

Debido a que Lua es el lenguaje de script de *Diököl* las aplicaciones deben ser escritas en este lenguaje. Sin embargo, la estructura de estos programas es muy similar a Processing, aún cuando, el manejo de eventos cambia ligeramente, tal como se puede observar en el siguiente segmento de código.

```
local mouseX=0
local mouseY=0
function setup()
  size(200,200)
end
function draw()
  background(255)
  stroke(0)
  fill(mouseX,0,mouseY)
  ellipse(mouseX-19,mouseY-30,16,32)
  ellipse(mouseX+19,mouseY-30,16,32)
```

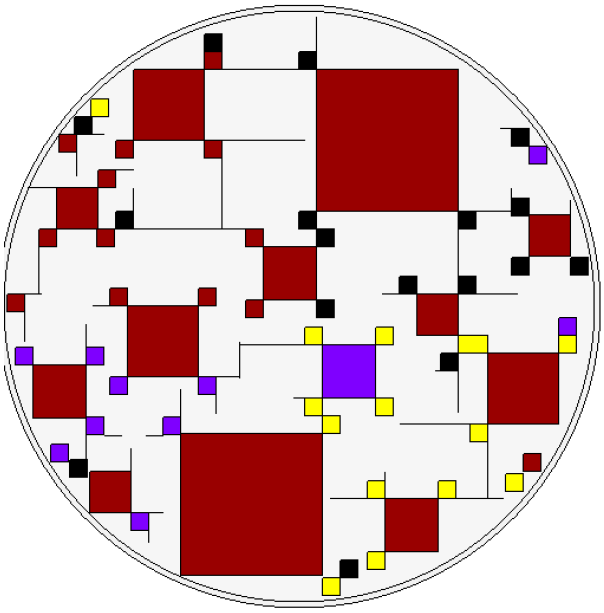

Fig. 5. Adaptación de una obra de Leon Polk Smith

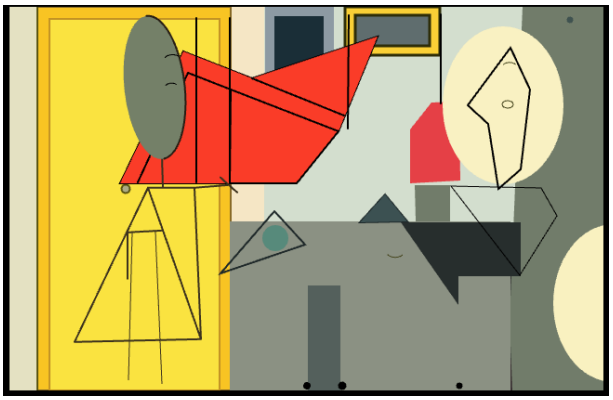

Fig. 6. Adaptación de L'Atelier de Pablo Picasso

```
end
function mouseMoved(x,y)
  mouseX = x
  mouseY = y
end
```

Esta forma de manejar los eventos generó algunos problemas a los estudiantes porque consideraron complejo mantener las posiciones del ratón utilizando variables. Otro aspecto complicado es determinar cuándo el cursor se encuentra sobre una figura, debido a que se deben realizar diversos cálculos geométricos que vuelven intrincado el código de la aplicación. Lo óptimo sería que el ambiente se encargara del manejo de eventos de forma interna y notificara a la aplicación cuando dicho evento ocurre, tal como se hace en SVG y HTML mediante la instrucción "onEvent".

D. Enfoque para manejo de eventos en Diököl

En vista de los inconvenientes que se presentaron con Diököl, posteriormente se hizo una modificación que incorporó nuevos métodos para el manejo de eventos (distintos a los que presenta Processing). El fin de esta actualización es evitar que los estudiantes se distraigan en escribir el código para detectar y manejar eventos, y se concentren en resolver el problema asignado.

Una aplicación práctica de este nuevo tipo de métodos para el manejo de eventos se realizó en el curso de Sistemas de Información Geográfica que se imparte en la carrera de computación. La creación y manipulación computarizada de mapas parece llamar la atención de los estudiantes porque un gran número matriculan esta optativa cada semestre. La incorporación de Diököl a este curso ha facilitado que los estudiantes diseñen aplicaciones complejas de visualización de información georreferenciada con poco código.

En versiones anteriores del curso se utilizaron otras herramientas para desarrollar aplicaciones geográficas (Leaflet, GoogleMaps, OpenLayers). Sin embargo, generalmente la curva de aprendizaje de esas bibliotecas era bastante alta y requería que los estudiantes dedicaran demasiado tiempo a programar funcionalidades muy básicas.

El manejo simplificado de eventos en Diököl es sencillo de aprender porque cada figura puede verificar la ocurrencia de un evento del cursor sobre ella. Para realizar esto, sólo es necesario especificar el tipo de evento que cada figura debe "escuchar" y en el momento que este se produce, la figura retorna un objeto asociado con la información generada. Este esquema facilita escribir aplicaciones "limpias" debido a que las instrucciones de detección del ratón sobre las diferentes figuras quedan ocultas del programador. Es importante indicar que este no es un procesamiento trivial, debido a que el ambiente utiliza el modo directo. Para ilustrar lo anterior, a continuación se muestra una adaptación en Diököl de un ejemplo tomado del libro "Visualizing Data" de Ben Fry [5]:

```
local csv = require("csv")
local mapImage
local locations
local randoms = {}
function setup()
  size(640,400)
  local font = createFont("Vera.ttf",12)
  textFont(font)
  mapImage = loadImage("data/map.png")
  locations = csv.open("data/locations.tsv")
  local data = csv.open("data/randoms.tsv")
  for fields in data:lines() do
    randoms[fields[1]]=fields[2]
  end
end
function draw()
  background(255)
  image(mapImage,0,0)
  event(MOVED)
  for fields in locations:lines() do
    local x = fields[2]
    local y = fields[3]
    local v = tonumber(randoms[fields[1]])
    if (v<0) then
      fill("#FF0000")
    else
      fill("#0000FF")
```

```
      end
      local evt = ellipse(x,y,v*4,v*4)
      if evt then
        fill(0)
        text(fields[1],x+5,y-5)
      end
    end
  end
```

La figura 7 permite observar que cuando se pasa el cursor sobre un estado de los Estados Unidos, se presenta la etiqueta del este (TX) utilizando la aplicación anterior.

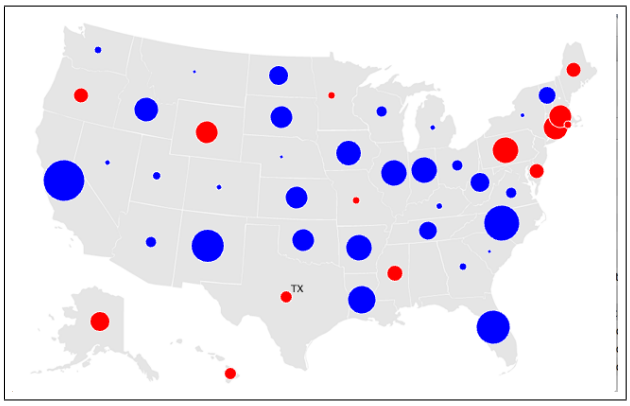

Fig. 7. Mapa interactivo

CONCLUSIONES

Aprender a programar es una tarea compleja que puede resultar frustrante y agotadora si no se cuenta con la motivación para dedicar un gran número de horas a aprender sobre métodos, estructuras y algoritmos. Por esa razón se han planteado diversas estrategias para facilitar el proceso de enseñanza y aprendizaje de forma contextualizada. En línea con esto, en este trabajo se presentaron las experiencias de la aplicación de diferentes aproximaciones para la enseñanza en cursos de programación de varias carreras usando gráficos interactivos.

La selección de las estrategias para la contextualización, así como de las herramientas que se utilizaran son un elemento clave para lograr los objetivos que se persiguen. Además, es importante tener en cuenta los conocimientos previos de los estudiantes y su disposición para aprender con base en sus intereses. Sobre este particular, los autores consideran que la selección de estrategias y herramientas que realizaron cuando comenzaron con la iniciativa no fue apropiada.

El método utilizado para generar gráficos interactivos desde un lenguaje de programación a archivos SVG era complejo y poco atractivo para los estudiantes, quienes no eran estudiantes de computación y solo habían recibido un curso básico de programación en Python, que era un lenguaje diferente al utilizado en el curso (C++), por lo que no se obtuvieron los beneficios que se perseguían. A lo que se debe agregar que la dificultad para definir la interacción con las figuras de forma sencilla en SVG hizo que el enfoque perdiera validez como un método para incentivar a los estudiantes.

Por su parte los ambientes de gráficos, como *VGDisplay*, que únicamente cuentan con comandos de dibujo y captura de eventos, pero no incluyen estructuras de control y manejo de ciclos pueden provocar que la interacción con el lenguaje base resulte lento o poco práctico. Lo que resulta más crítico cuando se realizan animaciones o arrastran objetos en la pantalla.

Como parte de las experiencias en el uso de gráficos interactivos con diferentes herramientas se determinó que en la enseñanza de conceptos elementales de programación un ambiente basado en un modelo de objetos (modo retenido) no resulta tan conveniente como hacerlo en uno basado en un conjunto de funciones simples (modo directo). Además, los ambientes gráficos basados en objetos parecen ser más convenientes para enseñar conceptos de composición y herencia a estudiantes de niveles superiores, porque ya conocen y dominan el paradigma de programación orientada a objetos.

Otro aspecto que no se tuvo en consideración es que los estudiantes requieren de tiempo adicional para conocer el ambiente gráfico y su funcionamiento. Lo cual provocó que en las clases se tuviera que invertir tiempo para explicar el funcionamiento de la biblioteca o ambiente de gráfico, y los conceptos básicos para la generación de gráficos (tipos de figuras, propiedades visuales, transformaciones geométricas). Esto implica que entre más sencillo sea el manejo del ambiente gráfico, más sencillo será para el estudiante entender su funcionamiento y empezar a utilizarlo.

El éxito de una estrategia de contextualización usando gráficos interactivos en un curso de programación depende de la sencillez para crear gráficos, el manejo de eventos y la facilidad de uso del ambiente. Si estos elementos son complicados o tediosos, los estudiantes no estarán motivados para utilizar la herramienta o programar funciones adicionales en una aplicación. En síntesis, lo ideal es contar con herramientas de programación de gráficos interactivos que cuenten con un conjunto de comandos simples de dibujo, como Processing, y que el mecanismo de manejo de eventos también sea simple, al estilo de SVG.

AGRADECIMIENTOS

Los autores ofrecen un agradecimiento especial al Centro de Investigaciones en Computación (CIC) de la Escuela de Computación por el apoyo ofrecido a esta investigación. Además, queremos agradecer a los estudiantes Erick Aguilar, Giancarlo Vargas, Adrian Salas, Diana Arce y Kevin Piedra por facilitar algunos de sus trabajos como ejemplos para este artículo.

REFERENCIAS

[1] Lecia J Barker, Charlie McDowell, and Kimberly Kalahar. Exploring Factors That Influence Computer Science Introductory Course Students to Persist in the Major. *SIGCSE Bull.*, 41(1):153–157, 2009.

[2] Jens Bennedsen and Michael E Caspersen. Abstraction Ability As an Indicator of Success for Learning Object-oriented Programming? *SIGCSE Bull.*, 38(2):39–43, 2006.

[3] Steve Cooper and Steve Cunningham. Teaching Computer Science in Context. *ACM Inroads*, 1(1):5–8, 2010.

[4] Natalie J Coull and Ishbel M M Duncan. Emergent Requirements for Supporting Introductory Programming. *Innov. Teach. Learn. Inf. Comput. Sci.*, 10(1):78–85, 2011.

[5] Ben Fry. *Visualizing Data*. 1st edition, 2008.

[6] D Ginat. On Novice Loop Boundaries and Range Conceptions. *Comput. Sci. Educ.*, 14(3):165–181, 2004.

[7] Ian Graham, Alan O'Callaghan, and Alan Cameron Wills. *Object-oriented methods: principles & practice*, volume 6. Addison-Wesley Harlow, UK, 2001.

[8] Mark Guzdial. A Media Computation Course for Non-majors. In *Proc. 8th Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, ITiCSE '03, pages 104–108, New York, NY, USA, 2003. ACM.

[9] Mark Guzdial. Does Contextualized Computing Education Help? *ACM Inroads*, 1(4):4–6, 2010.

[10] Craig Jenkins. A Work in Progress Paper: Evaluating a Microworlds-based Learning Approach for Developing Literacy and Computational Thinking in Cross-curricular Contexts. In *Proc. Work. Prim. Second. Comput. Educ.*, WiPSCE '15, pages 61–64, New York, NY, USA, 2015. ACM.

[11] Caitlin Kelleher and Randy Pausch. Using Storytelling to Motivate Programming. *Commun. ACM*, 50(7):58–64, 2007.

[12] Deepak Kumar and Lisa Meeden. A Robot Laboratory for Teaching Artificial Intelligence. In *Proc. Twenty-ninth SIGCSE Tech. Symp. Comput. Sci. Educ.*, SIGCSE '98, pages 341–344, New York, NY, USA, 1998. ACM.

[13] D Midian Kurland, Roy D Pea, Catherine Clement, and Ronald Mawby. A Study of the Development of Programming Ability and Thinking Skills in High School Students. *J. Educ. Comput. Res.*, 2(4):429–458, 1986.

[14] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. *ACM SIGCSE Bull.*, 37(3):14, 2005.

[15] Scott Leutenegger and Jeffrey Edgington. A Games First Approach to Teaching Introductory Programming. *SIGCSE Bull.*, 39(1):115–118, 2007.

[16] Allison Liu, Jeff Newsom, Chris Schunn, and Robin Shoop. Students learn programming faster through robotic simulation. *Tech Dir.*, 2013.

[17] Aleksi Lukkarinen and Juha Sorva. Classifying the Tools of Contextualized Programming Education and Forms of Media Computation. In *Proc. 16th Koli Call. Int. Conf. Comput. Educ. Res.*, Koli Calling '16, pages 51–60, New York, NY, USA, 2016. ACM.

[18] Rui Madeira and Dawid Gorny. *Cinder Creative Coding Cookbook*. Packt Publishing, 2013.

[19] John Maeda. *Design by Numbers*. MIT Press, Cambridge, MA, USA, 1999.

[20] L McCarthy, C Reas, and B Fry. *Getting Started with P5.js: Making Interactive Graphics in JavaScript and Processing*. Maker Media, Incorporated, 2015.

[21] Ibrahim Ouahbi, Fatiha Kaddari, Hassane Darhmaoui, Abdelrhani Elachqar, and Soufiane Lahmine. Learning Basic Programming Concepts by Creating Games with Scratch Programming Environment. *Procedia - Soc. Behav. Sci.*, 191:1479–1482, 2015.

[22] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, NY, USA, 1980.

[23] Denis Perevalov and Igor (Sodazot) Tatarnikov. *openFrameworks Essentials*. Packt Publishing, 2015.

[24] Semmy Purewal. Introductory Programming Concepts with Processing.Js. *J. Comput. Sci. Coll.*, 29(2):199–202, 2013.

[25] Richard Rasala. Toolkits in First Year Computer Science: A Pedagogical Imperative. *SIGCSE Bull.*, 32(1):185–191, 2000.

[26] Casey Reas and Ben Fry. *Processing: A Programming Handbook for Visual Designers and Artists*. The MIT Press, 2014.

[27] Eric Roberts and Keith Schwarz. A portable graphics library for introductory CS. In *Proc. 18th ACM Conf. Innov. Technol. Comput. Sci. Educ. - ITiCSE '13*, 2013.

[28] Eric S. Roberts. A C-based graphics library for CS1. *ACM SIGCSE Bull.*, 1995.