VIEWPOINTS OF PROGRAMMING DIDACTICS AT A WEB GAME IMPLEMENTATION

HORVÁTH, Győző – MENYHÁRT, László – ZSAKÓ, László, HU

Abstract: This article is written to those methodology specialist and practicing teachers who are paying attention to programming and who would like to make their class special. We present modern browsers' HTML5 and JavaScript support with a simple graphical game implementation. We make systematic programming enjoyable to our students. After understanding the basics, we present some game framework implemented in JavaScript, too, with which we can achieve spectacular results easier without giving up programming.

Keywords: graphics, programming, game, web, HTML5, JavaScript, frameworks.

1 Introduction

Web platform is good for teaching programming as it is shown in a lot of examples [6,7,8]. We have already dealt the connection between programming and web environment, so we have examined the possibility of using JavaScript as the first high level programming language [9,10]. Now we are building together these areas and looking for the chance of getting knowledge with simple but spectacular game implementation. After Scratch or Blockly graphical algorithm representation, so when the „small school" is finished it is time to write source codes. We introduce the new knowledges with mathematical problems in classical programming teaching. But at the same time students' maturity level does not gain this abstract and unconcerned level forced to the fictional data processing, so they lose their interesting very soon. Moreover there are very few students who are interested in maths, so we should build to other task groups.

The students in the grammar school are enough prepared to do some own graphical and image processing operation because they have already acquired the skills of elementary coordinate geometry on their maths courses. This presented task helps these knowledges' better deepening and maintains student's interest of informatics with a spectacular programming example.

First part of our article is moving towards the implementation of our idea from the systematic programming. Second part presents the necessary information about the environment and the JavaScript programming language. Finally we examine some game frameworks perspective of usability.

2 Methodological considerations and algorithm patterns

Learning programming is traditionally based on problems of mathematics and number theory, in which divisibility and prime numbers are in focus. This approach is suitable students who are interested in mathematics, moreover likes and can solve mathematical problems. But such students are very rare unfortunately. There are lots of attempts long ago to find an alternative task group.

In some cases the authors change only the range of tasks to be solved, such approach is the lecture notes written by Zsakó and Szlávi for biologist university students [1]. Here the basic operations are still reading from the input, writing to the output and variable assignment, which are organized into more difficult programs by adding conditional and loop statements.

Seymour Papert followed a different way [2], when he changed the scope of basic operations into commands of turtle graphics and text manipulation. Teaching the turtle graphics, loops precede conditionals, while in the case of text manipulations recursion appears instead of loops.

The theoretical basis of this article follows the latter approach, namely, let us change the scope of basic operations! Drawing a point to the screen will be our basic activity beside the variable assignment [3,4].

The basic idea is to move a point in a screen with fixed width and height. Initially the point is situated on the (c,r) coordinate. In order to understand the algorithms more easily, the dimensions of the screen will be 1024 columns and 768 rows.

As a first task move the point to the right with 100 units. This can be achieved with a count loop as an organizing tool!

```
Forward(N):
  r:=200; c:=200
  Repeat N times
    Point(c,r); c:=c+1
  End repeat
End procedure.
```

Second, move the point to the right border of the screen; the conditional loop is shown up as an organizing tool!

```
ToTheBorder(r,c):
  While c<1024
    Point(c,r); c:=c+1
  End while
End procedure.
```

It can be seen that with both movement we drew a line. But we have the opportunity to draw the point with an exclusive or operation by its colour. The nature of the exclusive or is that applying twice the original value is recovered, i.e. A XOR B XOR B=A. With this our original drawing becomes an animation.

The following procedure animates a point on the screen (applying a short waiting):

```
ToTheBorder(r,c):
  While c<1024
    Point(c,r); c:=c+1; Point(c,r); Wait
  End while
End procedure.
```

Next, give an arbitrary direction (dc, dr) to the point, and move it to the border of the screen. This is the selection algorithm, as we return the coordinate where the point stops moving.

```
ToTheBorder(r,c,dr,dc):
  While c>0 and c<1024 and r>0 and r<768
    Point(c,r); c:=c+dc; r:=r+dr; Point(c,r); Wait
  End while
End procedure.
```

Assume that there are lines drawn on the screen (walls). Stop the motion if the point reaches a wall. This is the linear search programming theorem detecting the colour of the points of the screen.

```
ToTheBorderOrWall(r,c,dr,dc):
  While c>0 and c<1024 and r>0 and r<768 and Co-
lor(c,r)≠background
```

```
    Point(c,r); c:=c+dc; r:=r+dr; Point(c,r); Wait
  End while
End procedure.
```

Make the point bounce back from the edge of the screen (for infinite times)! We need to introduce the conditional statement to achieve this goal.

```
BouncingBack(r,c,dr,dc):
  Loop
    Point(c,r); c:=c+dc; r:=r+dr; Point(c,r); Wait
    If c>1023 then c:=1023-(c-1023); dc:=-dc
    If c<0    then c:=-c;            dc:=-dc
    If r>767  then r:=767-(r-767);   dr:=-dr
    If r<0    then r:=-r;            dr:=-dr
  End loop
End procedure.
```

Let the point bounce back from the edge of screen, but if it collides to an inner wall, then stop its motion.

```
BouncingBackUntilWall(r,c,dr,dc):
  While Color(c,r)≠background
    Point(c,r); c:=c+dc; r:=r+dr; Point(c,r); Wait
    If c>1023 then c:=1023-(c-1023); dc:=-dc
    If c<0    then c:=-c;            dc:=-dc
    If r>767  then r:=767-(r-767);   dr:=-dr
    If r<0    then r:=-r;            dr:=-dr
  End while
End procedure.
```

Let the point bounce back from the edge of the screen regularly, and gradually slow down. Slowing down can be achieved by multiplying the distance travelled per time unit (i.e. velocity) by a f factor in every step. If f<1, then it is deceleration, if f>1, then it is acceleration.

```
BouncingBackWithDeceleration(r,c,dr,dc,f):
  Loop
    Point(c,r); c:=c+dc; r:=r+dr; Point(c,r); Wait
    If c>1023 then c:=1023-(c-1023); dc:=-dc
    If c<0    then c:=-c;            dc:=-dc
    If r>767  then r:=767-(r-767);   dr:=-dr
    If r<0    then r:=-r;            dr:=-dr
    dr:=f*dr; dc:=f*dc
  End loop
End procedure.
```

Let the point regularly bounce back from the edge of the screen, downwards accelerate, upwards decelerate. With this the physical knowledge can be presented in the program. According to the special coordinate system on the screen, downwards means the increasing screen row index.

```
BouncingInGravity(r,c,dr,dc):
  Loop
    Point(c,r); c:=c+dc; r:=r+dr; Point(c,r); Wait
    If c>1023 then c:=1023-(c-1023); dc:=-dc
    If c<0    then c:=-c;            dc:=-dc
    If r>767  then r:=767-(r-767);   dr:=-dr
    If r<0    then r:=-r;            dr:=-dr
    If dr≥0 then dr:=dr+1 else dr:=dr-1
  End loop
End procedure.
```

We get sufficient tasks for evolving the algorithmic structures. These spectacular tasks may be developed into games. We can find the place in them for other programming theorems: counting (counting the number of bounce backs or traversion through walls), filtering (colouring the walls already reached).

These algorithms can be implemented with simple, traditional programming languages, but they are not so amusing and spectacular compared to modern tools and intelligent interfaces. In the following these algorithms will help us developing the notorious brick game with HTML5 and JavaScript as a browser-based application.

3 Presentation of basics and environment

Students knows browsers, at least using them. We would like to use this platform-independent environment to create graphical and spectacular applications with writing source codes but without installing compilers and runtime environments. Native usage of graphical elements on the websites is possible with appearing of HTML5 and `canvas` element. It is supported by all current modern browsers.

The `canvas` gives an opportunity to manage two dimensional („`2d`”) and three dimensional („`webgl`”) contexts. Now we use `2d`, namely we create, modify and use a picture. The next sample source code draws a yellow filled circle into a black background picture (rectangle). A `canvas` tag is required in the `body` and there is a JavaScript function (`main`), what will run after the page load. Here getting 2d context of canvas and calling drawing function (`draw`) is located. The concrete rectangle drawing function is `fillRect`, but draw a circle and fill it must be coded in two different commands (`arc`, `fill`). More detailed descriptions can be found on a lot of websites like [11], so we did not specify these here.

```
<!DOCTYPE html><title>Example</title>
<canvas id="canvas" width="1024" height="768"></canvas>
<script type="text/javascript">
  var canvas = document.getElementById('canvas');
  var ctx = canvas.getContext('2d');
  function draw() {
    //Clear
    ctx.fillStyle = 'black';
    ctx.fillRect(0, 0, canvas.width, canvas.height);
    //Ball
    ctx.fillStyle = 'yellow';
    ctx.beginPath();
    ctx.arc(512, 384, 50, 0, Math.PI*2, true);
    ctx.closePath();
    ctx.fill();
  }
  draw();
</script>
```

In case of visualising movements, we have to redraw the picture sometimes. Redrawing the object is not enough as it mentioned in the previous section, because the earlier shape stays there as well. Function `XOR` is effective for simple, two colours figures, but we use the simplest and general erasing the whole picture method in the next example.

This drawing is enough resource intensive so we do not use the simple function `setInterval` to show the figure. We do not use the function `setTimeout` with which the time of the next draw should be defined, but we apply the function `requestAnimation-`

Frame what uses better performance of browsers [12]. It is similar to setTimeout, but with this the function gameloop is run by the browser at refresh rate. Because of this the time intervals can be different at each drawing so we have to calculate the time difference between the two drawing and calculate the position of the objects in function update and draw it to the new position in function draw.

We have to correct the algorithms above during the coding because of restriction of technology namely they took advantage of having same interval. Function main must be modified as it has an object (body) for the ball and a variable (prevT) to save the timestamp so we will be able to calculate the elapsed time. Function update calculates the new position of the ball by time. In function draw we use the actual position values of the body to draw it.

```
var body = {
  x: 512,  y: 384, r: 50,  // px
  vx: 100, vy: 60 // px/s
};
var prevT = Date.now();
function gameloop() {
  var t = Date.now();
  var dt = t - prevT;
  prevT = t;
  window.requestAnimationFrame(gameloop);
  update(dt);
  draw();
}
function update(dt) {
  body.x += body.vx * dt/1000;
  body.y += body.vy * dt/1000;
  var frame = {
    bfx: body.r, jax: canvas.width - body.r,
    bfy: body.r, jay: canvas.height - body.r
  }
  if (body.x > frame.jax) {
    body.x = frame.jax - (body.x - frame.jax);
    body.vx = -body.vx;
  }
  if (body.x < frame.bfx) {
    body.x = frame.bfx + (frame.bfx - body.x);
    body.vx = -body.vx;
  }
  if (body.y > frame.jay) {
    body.y = frame.jay-(body.y- frame.jay);
    body.vy = -body.vy;
  }
  if (body.y < frame.bfy) {
    body.y = frame.bfy + (frame.bfy - body.y);
    body.vy = -body.vy;
  }
  body.vy += 1;
}
function draw() {
  //Clear
  ctx.fillStyle = 'black';
  ctx.fillRect(0, 0, canvas.width, canvas.height);
  //Ball
  ctx.fillStyle = 'yellow';
  ctx.beginPath();
  ctx.arc(body.x, body.y, body.r, 0, Math.PI*2, true);
  ctx.closePath();
  ctx.fill();
}
gameloop();
```

From now the implementation of first version of notorious brick game is very easy because the previous elements must be used only. There will be a list of objects `bricks` besides the object ball (`body`). We have to modify the function `update` to rebound the ball at the bricks not only at the edges. But they have to be deleted from the list at this time. Function `draw` must show all objects from this list.

In the next step an object `bumper` is declared with its size and position. The ball must be reflected from this, too, but it is not deleted and we should move it horizontal with the mouse or keyboard. Here we meet with a newer programming technology concept, the event control. The browser listens to every key press, we add it with function `addEvent-Listener`, and it triggers a function `moving`. So we will be able to modify the position of object `bumper` with this. We have to add the following extension to the JavaScript code only:

```
var bumper = {
  x: 512,  y: 758, w: 100,  h: 20,  // px
  moving: 25 // px
}
function moving(event) {
  if (event.keyCode==37) {
    //moving left
    if (bumper.x-bumper.moving-(bumper.w/2)>0) {
      bumper.x=bumper.x-bumper.moving;
    } else {
      bumper.x=bumper.w/2;
    }
  } else if (event.keyCode==39) {
    //moving right
    if (bumper.x+bumper.moving+(bumper.w/2)<canvas.width){
      bumper.x=bumper.x+bumper.moving;
    } else {
      bumper.x=canvas.width-(bumper.w/2);
    }
  }
}
document.addEventListener("keydown",moving,true);
```

After these modifications reflections must be handled in function update and our own brand game is completed.

4 Using game engines

In the previous chapter we could see that, beside the `update` function containing the core logic, numerous other code fragments needed to display and run our application in the browser. Although these code fragments are necessary parts of the whole code base, still these program codes are simpler, more schematic, requiring less thinking and creativity, compared to the data description and the core logic parts of the program. During education it is worth drawing the students' attention to these repetitive and schematic code fragments, and make them reusable by rearranging them as parametric functions. With this rearrangement we can take one step forward on the abstraction scale, because we could pull out such parts of the whole monolithic code base, which are functionally unified subprograms. In our application the `gameloop` function, the drawings of the different objects, counting the elapsed time between two cycles, updating the different aspects of the state-space (moving the ball, handling collisions) are some possible examples where such abstraction could be made.

While the above abstraction is related to the decomposition of an imperative program, handling the entities of the game leads us to another important programming paradigm. Although it was necessary to store and update the position of the ball in the naïve solution, the program did not take care of the functional integrity of these code parts. The next possible step in the abstraction scale is the identification of the different entities in the application, and ensuring the encapsulation of the cohesive data and procedures. In this way data representation and the object-oriented paradigm can be introduced in a very intuitive way during analysing the task. In our example we can identify the objects of the imagined game field and can think about their data members and methods.

The next step might be the embedding of these repetitive and reusable code fragments into a uniform environment which provides the frames of the software development. In this abstraction layer predefined conceptions and utility functions help developing the solution. These software environments are called software frameworks. Frameworks have two fundamental basic purposes: on the one hand they have a very definite *conception* about the process of the development, the right places of specific code fragments, and the main philosophy of the program operation; on the other hand they provide best practices and *ready-made solutions* for recurrent problems.

In the field of web-based game development almost every game framework provides solutions for the game loop, for the calculation of the elapsed time between two cycles, for the way and technology of rendering, for defining the game objects and observing their interactions, for resource management (e.g. image, sound effects, animations), and other characteristics which change from frameworks to frameworks. Every framework has its own specific conception about code management as well: some absolute modular, others more monolithical, there are more declarative or more imperative approaches among them.

The actual implementation depends obviously on the selected framework. In this article the Frozen game framework [5] is used for demonstration, not because of its wide reputation, but it recognizably shows the naïve starting point and the different abstraction layers mentioned above are also easily can be applied. Game preparation, configuration and the game loop takes place in the `GameCore` object, which can be started with the `run` method after instantiation and parameterization:

```
var game = new GameCore({
  canvasId:         'canvas',
  gameAreaId:       'gameArea',
  canvasPercentage: 0.95,
  update:           update,
  draw:             draw
});
game.run();
```

The logic of the bouncing ball is located in a separate module, where the data describing the ball and the related operations (e.g. the `update` method) are encapsulated. Also this module is responsible for rendering the ball to the canvas (`draw` method):

```
function Ball() {
  this.x = 50; this.y = 20; this.r = 5;
  this.vx = 100; this.vy = 60;
  this.update = function (millis) {
    //Similar to the previous update method
  };
  this.draw = function (ctx) {
    //Similar to the previous drawing
  };
}
```

With this solution the update and draw phase of the game loop simply consists of the calling of the `update` and `draw` methods of the ball object.

The motion and interaction of objects usually follows real-world mechanisms. In the case of such games the implementation of these physical laws can be carried out by another abstract layer: the physics engine. These physics engines also work with objects with position and velocity, but beside these they have other real-world properties, such as mass, momentum and angular momentum, shape and material. These objects can interact and collide with each other while moving in a gravitational field. Physics frameworks often simplify the creation and movement of objects by parameterization: instead defining object behaviours the parameterization and population of the virtual world is in focus.

From the numerous physics engine (p2 [13], box2d [14]) the Frozen framework ensures deeper integration with the box2d physics engine through the `BoxGame` class. The box2d objects added to the instance of this class are automatically handled, moved and drawn by the Frozen framework. The following code example shows how the `GameCore` class expands with the box data member (representing the `box2d` features), and how a new ball can be added to the virtual world by the `addNewBall` method:

```
var game = new BoxGame({
  //Earlier parameters
  //...
  box: new Box({gravityY: 0}),
  addNewBall: function () {
    var ball = new Ball();
    game.addBody(ball);
    game.box.applyImpulseDegrees(ball.id,155,
                                 ball.impulse*0.75);
  }
});
game.addNewBall();
game.run();
```

For the full example only the operational objects must be added to the physical world: the four walls and the bouncing ball. The ball class extends the Circle class of the box2d library, inheriting the properties of circle-shaped physical objects. (Inheritance is provided through the dcl library.) Movement and drawing is guaranteed by the framework, drawing can be customized if necessary. The perfectly elastic bouncing ball is simulated with the following parameters:

```
var Ball = dcl(Circle, {
  ball: true,
    x: 60, y: 210, radius: 10,
    restitution: 1.0, friction: 0, impulse: 5,
    constructor: function(){
      this.id = this.id || _.uniqueId();
    }
});
```

Examining the final solution, we can observe that at this level of abstraction the programming theorems are diminished. This can be strange, as our original purpose was the practice of these theorems through interesting programming tasks such as games. But those theorems are hidden behind abstract programming interfaces and libraries. Nevertheless, these programming theorems can also be introduced, with the objects of this higher-level abstraction. Programming theorems are defined on series. At this level of abstraction of games we have series of physical objects, entities, sprites, and so on. For the game logic not only their movements, but other examinations also take place, which necessarily involves the usage of programming theorems.

5 Conclusion

The environments and technologies presented in this article give possibilities to make spectacular and playful applications which can motivate students to learn programming. The way shown in this article leads the reader from the algorithmical thinking through the technological knowledge to the higher levels of abstraction and tools, and hopefully invites other teachers to give a try for this method with their students.

We definitely think that there are many ways to endear programming with students, and none of them can be exclusive. However, we are sure that the method of developing web-based games can be successful for a significant group: many students report that making a game is much more interesting than playing games.

During the planning of this article it was a definite aspect to make the systematic programming and programming theorems as the base, and to motivate students with the words of web environment and game development. During the implementation new technologies are introduced, and finally new abstraction layers can be carried in through the different frameworks.

Summing up, the example task in this article is capable to give a full answer to the criteria of didactics, such as purpose, content, process, organization, method and tools.

6 References

1. P. SZLÁVI, L. ZSAKÓ: *Bevezetés a számítástechnikába*. Egyetemi jegyzet. Tankönyvkiadó. (1987)
2. S. PAPERT: *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, Inc. New York, NY, USA (1980)
3. L. ZSAKÓ, P. SZLÁVI: *Grafikai alapalgoritmusok*. INF.O.'94 Informatika és számítástechnika tanárok konferenciája. Békéscsaba, Magyarország, 1994.11.17-1994.11.19. (1994)
4. L. ZSAKÓ, P. SZLÁVI: *Az informatika oktatása*. Budapest, ELTE Informatikai Kar. (2014)
5. http://frozenjs.com/docs/ (last visited: 2015.10.31.)
6. https://code.org/learn (last visited: 2015.10.31.)
7. https://blockly-games.appspot.com/ (last visited: 2015.10.31.)

8. https://www.khanacademy.org/computing/computer-programming (last visited: 2015.10.31.)
9. GY. HORVÁTH, L. G. MENYHÁRT: *Teaching introductory programming with Javascript in higher education* (Volume 1. pp. 339-350) DOI: 10.14794/ ICAI.9.2014.1.339 (2014)
10. GY. HORVÁTH, L. G. MENYHÁRT: *Oktatási környezetek vizsgálata a programozás tanításához*. Infodidact 2014, Zamárdi, 2014.11.23-24. http://infoera.hu/infoera2014/ea/infodidact2014_horvathgy_menyhartl.pdf (2014)
11. https://developer.mozilla.org/en-us/docs/web/api/canvas_api (last visited: 2015.10.31.)
12. https://developer.mozilla.org/en-us/docs/web/api/window/requestanimationframe (last visited: 2015.10.31.)
13. https://schteppe.github.io/p2.js/ (last visited: 2015.10.31.)
14. http://box2d-js.sourceforge.net/ (last visited: 2015.10.31.)

Lectured by: Gábor Törley, PhD.

Contact address:

Győző Horváth, Dr. Ph.D.,
Department of Media and Educational Informatics, Faculty of Informatics, Eötvös Loránd University, H-1117 Budapest, Pázmány P. sétány 1/C, Hungary,
phone: +36-1-372-2500 , e-mail: gyozke@elte.hu
László Menyhárt,
Department of Media and Educational Informatics, Faculty of Informatics, Eötvös Loránd University, H-1117 Budapest, Pázmány P. sétány 1/C, Hungary,
phone: +36-1-372-2500 , e-mail: menyhart@inf.elte.hu
László Zsakó, Dr. hab. Ph.D,
Department of Media and Educational Informatics, Faculty of Informatics, Eötvös Loránd University, H-1117 Budapest, Pázmány P. sétány 1/C, Hungary,
phone: +36-1-372-2500 , e-mail: zsako@caesar.elte.hu