УДК 004.92
DOI: 10.32347/0131-579x.2021.100.215-228

PhD, associate professor **Tregubova I.A.,**
pisma-irine@ukr.net, ORCID 0000-0003-2030-7678
itnek0pi@gmail.com, student **Hryhorashchenko V.O.**

State University of Intellectual Technologies and Communications, Odessa

IMPLICATION THE CROSS-PLATFORM LOVE2D ENGINE FOR RENDERING AND ARTIFICIAL INTELLEGENCE DEVELOPMENT

Recently, the market for consoles and mobile games is growing, and therefore to find a game engine that meets the demanding requirements of users is not an easy task. Technology platforms have become clear favorites of many developers. However, the market is volatile, and therefore the question of choosing a game engine will not lose its relevance in the near future and is the first, main, relevant and important subject of choice in this work. The relevance of this article is to write an original algorithm for 2D computer game, taking into account the latest intelligent technologies, which will be different from previous versions by its uniqueness: a new procedural generation, improved artificial intelligence of characters, original game mechanics, which necessitated the creation of a key to unlock levels.

To work on this task, the open cross-platform LOVE2D engine and the Lua programming language a powerful, efficient and easy to learn language were substantiated. The reason for choosing the LOVE2D game engine is that its technology is unique in itself. Simple text was used in the development of the game algorithm. A significant number of issues were resolved in unique way from scratch while developing.

Original positions of game mechanics are created such as unlocking the end of the level, the infinity of the game and focusing on the maximum "Score", increasing the size of each subsequent level, compared to the previous one, improved artificial intelligence of characters are the main differences from existing approaches used to create previous Super Mario Bros projects. The reason that makes this project more advanced and a little more random than the original game: it's a procedural level generation. Since the game is infinite it's really important to keep levels random and different.

The obtained results made it possible to say that the work done is a new step forward in comparison with previous developments of algorithms for this 2D game. The original algorithm and code for 2D computer game, using the capabilities of modern information technologies, can be useful not only for creating mobile games, but also for solving virtual reality, augmented reality, TV presentation, visualization effects of the hologram.

The project presented in the paper is made exclusively for the purpose of implementation in the educational process.

Keywords: cross-platform engine; programming language; procedural generation; algorithm, program code; rendering; artificial intelligence.

Problem statement. Every year more and more computer games are improved thanks to the fruitful work of professionals on the creation of 2D and 3D games.

The urgency of the problem lies in the fact that the created algorithm corresponds to the latest achievements of computer science and makes the game original, interesting and exciting for users. To do this, it was necessary to create a new original generation, improve the artificial intelligence of the heroes to facilitate the possibility of passing the levels.

Analysis of recent research and publications. In 1961, at the Massachusetts University of Technology, programmers at one of the mainframes created and launched the world's first prototype of a computer game. It was given the name Space War. The essence of the game was that two spacecraft plowed the open spaces of the monitors, and tried to hit each other with shells. The world's first computer game was made in the arcade genre.

Computer games went to the people 10 years after the events described above. The name of the arcade genre was born in 1971. Entrepreneur Nolan Bushnell invented and launched the first commercial arcade game. Both the principle and the plot remained the same as with Star Wars, but the apparatus on which all this was reproduced changed. It was an affordable simple iron box with a monitor that vaguely resembled modern slot machines.

The arcade genre prevailed for several years in a row, but in the mid-seventies the first adventure game was released, which was called Adventure. Further, in 1976 the first computer game console with one game was released, a year later – an analogue, but with several games.

At the beginning of the 80s of the last century, there was a boom in the release of game consoles.

From that moment on, everything was given exclusively to time. It went on, computers became more powerful, and games were brighter and more functional.

Having spread rapidly, virtual computer games have become an integral part of modern culture. Having become an element of the everyday life of thousands and millions of mainly young people around the world, they have formed new subcultures. Virtual games form new traditions and behavioral skills that change the structure of society.

Highlighting previously unresolved parts of a common problem. For the first time for a 2D computer game Super Mario Bros, the cross-platform Love 2D engine and the Lua programming language were chosen as the most rational option. At the same time, the positive and negative aspects of the most popular

2D and 3D game engines were considered and analyzed. As a result of the analysis, the above-mentioned game engine was selected for this case.

Objective. The aim was to write own original algorithm for the 2D computer game Super Mario Bros, which differs from previous versions by its uniqueness: a new procedural generation, improved artificial intelligence of characters, original game mechanics, which necessitated the creation of a key to unlock levels.

Presentation of the main material. To choose of an appropriate game engine and programming language for the Super Mario Bros game, an analysis was made of the positive and negative sides of a certain number of 2D and 3D graphics game engines, on which the gaming industry is usually based.

As a result of the analysis of the positive and negative aspects of game engines, the 2D game engine LÖVE 2D was chosen as the most rational for the chosen game. Programming languages for creating games are now more and more in demand and developed.

Programming language Lua was chosen to work in an article on the creation of the game.

LÖVE is a great, completely free, simple 2D game engine. LÖVE (or Love2D) is an open-source cross-platform engine for developing 2D computer games. Designed in C++, it uses Lua as a programming language. Free 2D game engine LÖVE is a wonderful framework that is convenient to use for creating 2D games in Lua. It works on Windows, Mac OS X, Linux, Android and iOS. To make a one-time customization of any game, Love2D(LÖVE), has been used for commercial projects, game plugs, prototypes, and everything in between. When writing a game using LÖVE, the most important parts of the API are the following callbacks:

love.load,
love.update,
love.draw
love.graphics.

Love.update is used to control the state of the game from frame to frame, and love.draw is used to display the state of the game on the screen. LÖVE creates its function with the same name as the callback calls. The primary responsibility for the love.graphics module is the drawing of lines, shapes, text, images and other drawable objects onto the screen. Its secondary responsibilities include loading external files (including images and fonts) into memory, creating specialized objects (such as Particle Systems or Canvases) and managing screen geometry.

Obvious cons are that it’s not visual and the set-up is not very user friendly. Other point is that it doesn’t feature physics system, the high-definition rendering, shaders system, particle systems, and so on, but it is redundant in this particular case.

What’s good about it as it encourages the programmer to go out to learn and think how to implement those features, and it gives programmer the boundaries that were met by many classical retro games creators except for the obvious performance and memory usage boundaries. That is why the authors show how much hard work was put into modern game engines and teach how to overcome those problems.

Lua is used for all sorts of applications, from games to web applications and image processing. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode with a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

Lua has a deserved reputation for performance. Lua as the fastest language in the realm of interpreted scripting languages. It’s fast not only in fine-tuned benchmark programs, but in real life too. Substantial fractions of large applications have been written in Lua. Lua also makes it easy to extend programs written in other languages. Lua has been used to extend programs written not only in C and C ++, but also in Java, C #, Smalltalk, Fortran, Ada, Erlang, and even other scripting languages such as Perl and Ruby.

Why Lua was a choice for beginning with programming in Gamedev:

Well documented: reference manual, book, wiki, 6-page quick reference and more.

A friendly and enthusiastic community. Thanks to excellent documentation, wiki, mailing list, and StackOverflow [1], authors had no problem finding answers to questions.

Clean and simple syntax suitable for beginners and accessible to non-programmers. Lua borrowed much of its control syntax from Modula, a descendant of Pascal, which was widely used in education as an introductory language.

Built-in interpreter: just run Lua from the command line.

An incremental garbage collector with low latency, no additional memory overhead, little implementation complexity, and support for weak tables.

Assigning a value to nil removes the element from the table. This is consistent with returning nil for a non-existent element, so it doesn't matter if the element does not exist or exists with a nil value. a = {b = nil} produces an empty table. This makes it easy to destroy game objects in Lua.

Indeed, Love2d is a very lightweight and powerful game engine. Most suitable game engine for retro games, such as the ones that the authors used to develop their game development skills

Things that are just different in Lua

Tables and strings are indexed from 1 rather than 0, “not”, “or”, “and” keywords used for logical operators, there are no a+=1, a+, or similar shorthand forms.

One of the challenges that the authors had to face is that Lua doesn’t have classes, therefore the object-orientation is implemented using tables and functions; inheritance is implemented using the meta-table mechanism. But for clarity, the following GitHub [2] utilities were used, called hump by Matthias Richter [3] for solving different issues such as input control, and class.lua [4] to add classes to lua.

Along the journey in the gamedev the authors also stumbled upon a problem when trying to set a virtual resolution on the screen and authors decided to use package by Ulysse Ramage [5] to make it more manageable.

Whilst LOVE2D has its own solutions for Timer and creating events, the authors turned to a “knife” package [6].

Procedural generation. "Mario Level-Maker" Algorithm pictured on figure 1.1 has the following steps of execution:

Variables initialization for storing info and cache reference about generated objects and to keep track of whether specific objects were placed once (such as pole and key block object).

Starts iterating through x coordinate up to the current width of the level (this parameter changes exponentially depending on the current level player is on).

Checking specified conditions to place objects (e.g. where to place emptiness, flat surfaces, gaps, jump blocks, bushes, key-block, end level pole).

When iteration has ended, returns cached generated objects to have access to then work with them outside of the level generation script.

Also of note is the reason that makes this project a little more advanced than the original game: it's a procedural level generation. Once again, this is a bunch of if / else statements that put objects on the stage following some simple / or not so big rules. This makes the developed project a little more random:

Usually, procedural generation is a combination of if/else conditions and randomizing. Since the game is infinite it's really important to keep levels random and different. Below is the little simplified version of the actual algorithm that is used in this project for procedural generation of the levels.

The article demonstrates the ease of use of the basics LOVE2D using an example developed by the authors of the Mario project.

Love 2D makes it easy to set variables and core parameters before rendering first frame in function love.load()

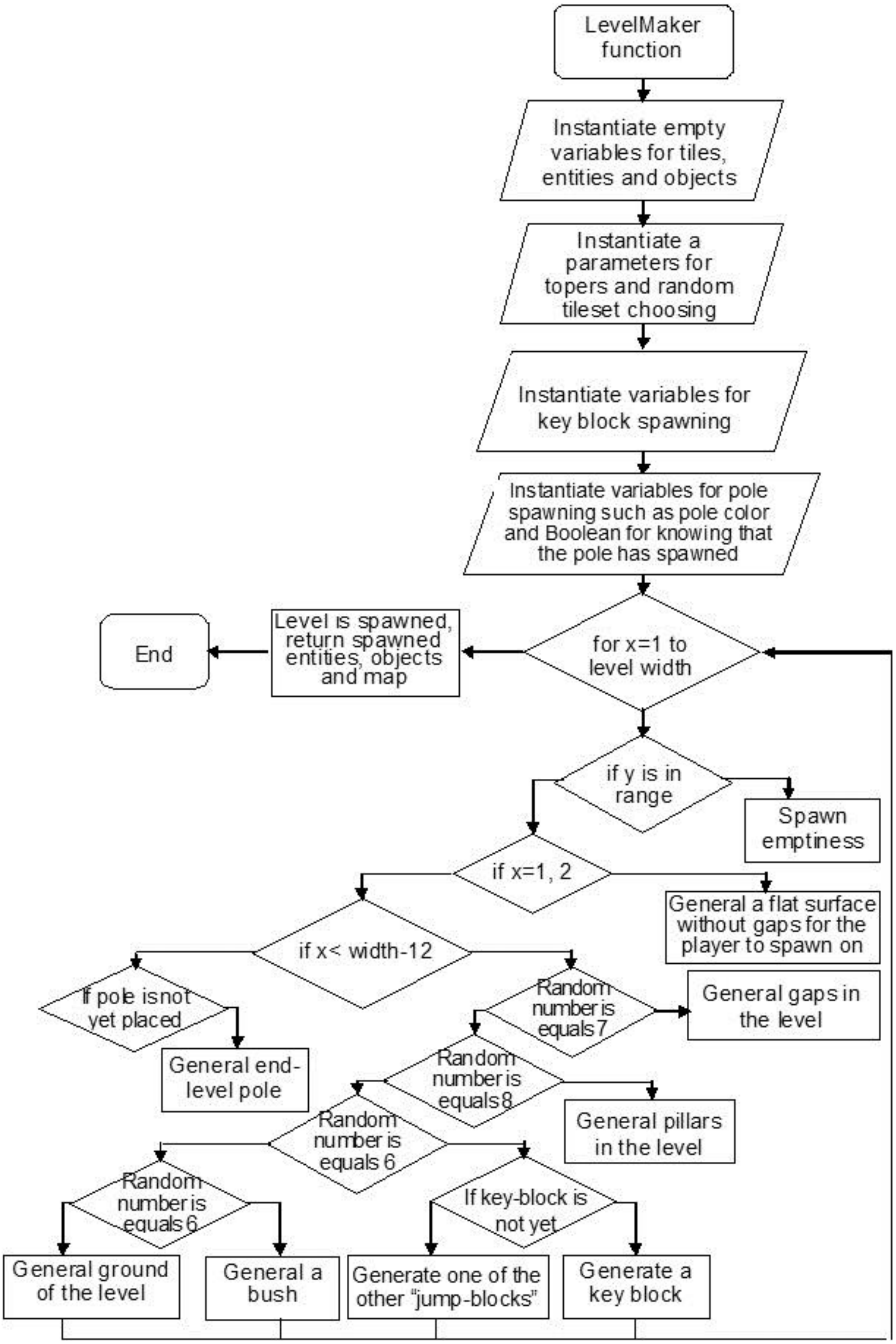
LevelMaker function
Instantiate empty variables for tiles, entities and objects
Instantiate a parameters for topers and random tileset choosing
Instantiate variables for key block spawning
Instantiate variables for pole spawning such as pole color and Boolean for knowing that the pole has spawned
for x=1 to level width
Level is spawned, return spawned entities, objects and map
End
if y is in range
Spawn emptiness
if x=1, 2
General a flat surface without gaps for the player to spawn on
if x< width-12
If pole is not yet placed
General end-level pole
Random number is equals 7
General gaps in the level
Random number is equals 8
General pillars in the level
Random number is equals 6
Random number is equals 6
If key-block is not yet
General ground of the level
General a bush
Generate one of the other "jump-blocks"
Generate a key block

Fig. 1.1 Procedural generation algorithm's flow-chart

```
function love.load()
    love.graphics.setDefaultFilter('nearest','nearest')
    love.graphics.setFont(gFonts['medium'])
    love.window.setTitle('Super 50 Bros - Nek0pi edition')
```

In this snippet, a default filter has been set. A font has been set to be used for a variable that was previously defined before it contains the path to the ".tff" file, and the title of the window has been set to "Super 50 Bros – Nek0pi edition".

So called nearest neighbor which allows to gain following effect as pictured in figures 1.2 and figure 1.3.

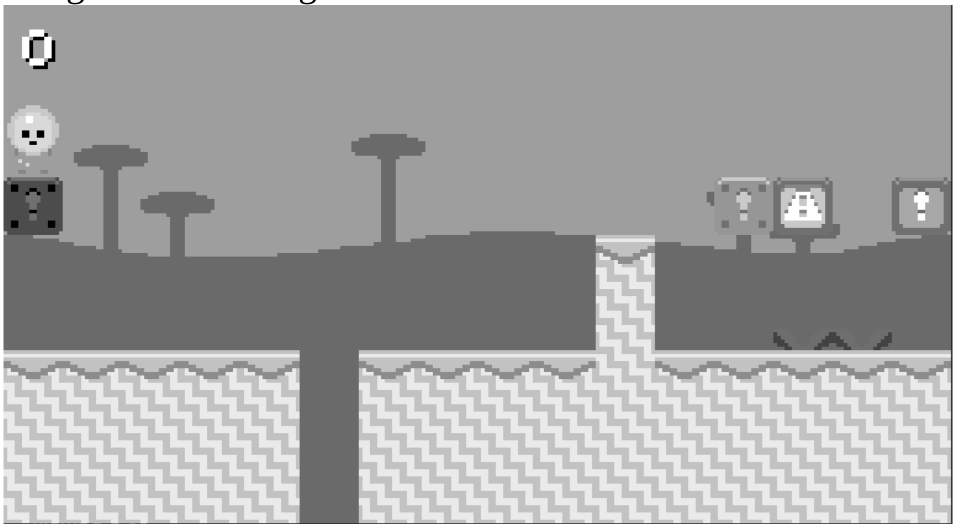

Fig. 1.2 Pixel-art graphical filter for nearest neighbor not applied

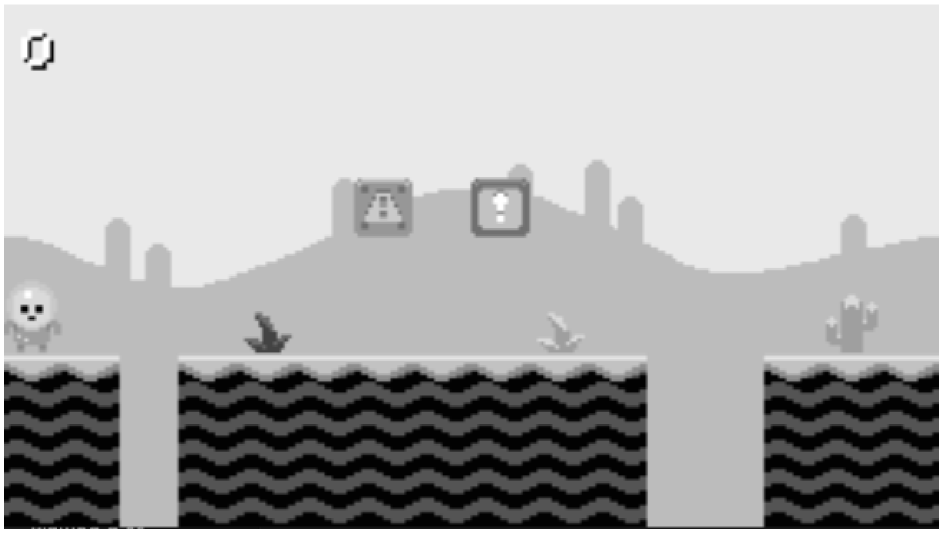

Fig. 1.3 Pixel-art graphical filter for nearest neighbor applied

As a default – love2d applies an Anisotropic Filtering to smooth out the sprites.

Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode with a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

The following shows that a push packet was used to set a virtual size of the screen, turn on Vsync and make the windows be resizable.

```
push:setupScreen(VIRTUAL_WIDTH,VIRTUAL_HEI
    GHT,WINDOW_WIDTH,WINDOW_HEIGHT, {
fullscreen = false,
vsync = true,
resizable = true
})
```

Also, the love.resize () function was used for the push package to work.

```
function love.resize(w, h)
    push:resize(w, h)
end
```

Next, Love2d provides a function to draw something on the screen every frame - love.draw().

For example, to see how the text is displayed on the screen, it's necessary to take this code from the main screen:

```
  -- Setting the font to use
    love.graphics.setFont(gFonts['title'])
    -- Setting the color to black to draw a shadow
    love.graphics.setColor(0, 0, 0, 255)
    -- Using love.graphics.printf to render Text, which is
placed on the center of the screen
    love.graphics.printf('Super      50     Bros.',     1,
VIRTUAL_HEIGHT / 2 - 40 + 1, VIRTUAL_WIDTH, 'center')
     -- Setting the color to white to draw text
    love.graphics.setColor(255, 255, 255, 255)
```

To see what each line does, it's recommended to refer to the comments in the code.

Now that the rendering is complete, it's apparent how the controls for the character are created.

In this regard, if no physics engine is involved, the programmer will have to come up with his own way to establish the laws of physics and define what a collision is. The authors created the Player class, which contains functions for checking collisions, and also implemented various states for the player to jump / fall / walk and idle.

The code is too big to attach as a picture. But here's a small snippet of code from the Walk Player state to get an idea of what's going on there.

If not love.keyboard.isDown('left') and not love.keyboard.isDown('right') then

```
            -- changes the player state to idle
            self.player:changeState('idle')
        else
            -- sets codinates of left and right bottom tiles
            local               tileBottomLeft               =
self.player.map:pointToTile(self.player.x + 1, self.player.y +
self.player.height)
            local              tileBottomRight               =
self.player.map:pointToTile(self.player.x + self.player.width -
1,
            self.player.y + self.player.height)

            -- temporarily shift player down a pixel to test
for game objects beneath
            self.player.y = self.player.y + 1

            local              collidedObjects               =
self.player:checkObjectCollisions()

            self.player.y = self.player.y – 1
```

To see what each line does, it’s recommended to refer to the comments in the code.

State control was used for the AI system as it maintains order and makes the code easy to read. Consider an example with a snail as the main enemy in the Mario project.

When talking about artificial intelligence in games, it usually means some if / else condition that makes enemies / NPCs behave in a certain way. The AI of the enemies in the original Super Mario game was not surprising, so an attempt was made to do something similar in spirit and the following was implemented:

All together there are 3 states: Chasing state, Idle State and Moving state.

Idle and moving states are there to imitate random walking of enemies to make them look more alive.

Chasing state is there to check conditions to know when to start moving towards the Player’s Main Character.

The code is pretty big, attaching snippets to get some idea of how it all works.

```
    function SnailChasingState:update(dt)
        self.snail.currentAnimation:update(dt)
```

```
     -- calculate difference between snail and player on X axis
     -- and only chase if <= 5 tiles
         local diffX = math.abs(self.player.x - self.snail.x)

         if diffX > 5 * TILE_SIZE then
             self.snail:changeState('moving')
         elseif self.player.x < self.snail.x then
             self.snail.direction = 'left'
             self.snail.x = self.snail.x - SNAIL_MOVE_SPEED* dt
     -- * stop the snail if there's a missing tile on the floor
to the left or a solid tile directly left
             local                  tileLeft                  =
self.tilemap:pointToTile(self.snail.x, self.snail.y)
             local               tileBottomLeft               =
self.tilemap:pointToTile(self.snail.x,      self.snail.y      +
self.snail.height)

             if    (tileLeft    and    tileBottomLeft)    and
(tileLeft:collidable() or not tileBottomLeft:collidable()) then
                 self.snail.x = self.snail.x + SNAIL_MOVE_SPEED
* dt
             end
             -- * If in range of player stop
             elseif  self.player.x  -  2  <  self.snail.x  and
self.player.x + 2 > self.snail.x then
                 self.snail.x = self.snail.x
             else
             self.snail.direction = 'right'
             self.snail.x = self.snail.x + SNAIL_MOVE_SPEED* dt
         end
     end
```

Identification of "gaps" has also been implemented so that snails do not fall into them, chasing the player.

```
     -- stop the snail if there's a missing tile on the floor
to the right or a solid tile directly right
             local                  tileRight                  =
self.tilemap:pointToTile(self.snail.x    +    self.snail.width,
self.snail.y)
             local               tileBottomRight               =
self.tilemap:pointToTile(self.snail.x    +    self.snail.width,
self.snail.y + self.snail.height)

     if   (tileRight   and   tileBottomRight)   and   (tileRight:
collidable() or not tileBottomRight:collidable()) then
             self.snail.x = self.snail.x - SNAIL_MOVE_SPEED* dt
```

```
        end
```

Please refer to figure 1.4 to see what issues where addressed by implementing the above code.

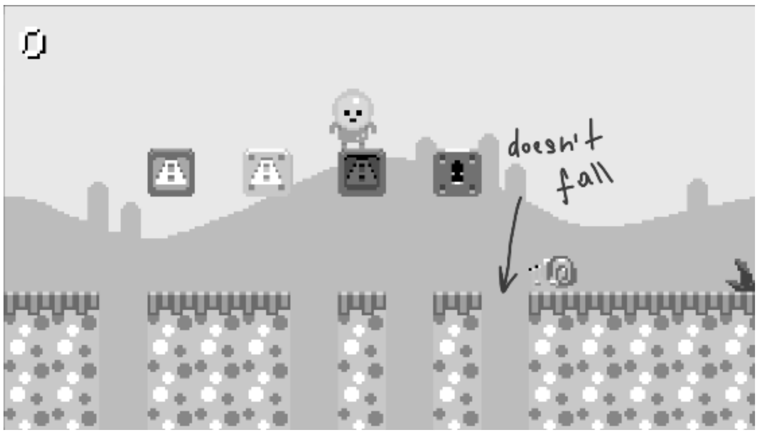

Fig 1.4 Gaps that were problematic for virtual enemies

Please refer to Figure 1.5 to see a good example of poor implementation of rules which can potentially lead to blocking the player from completing the game.

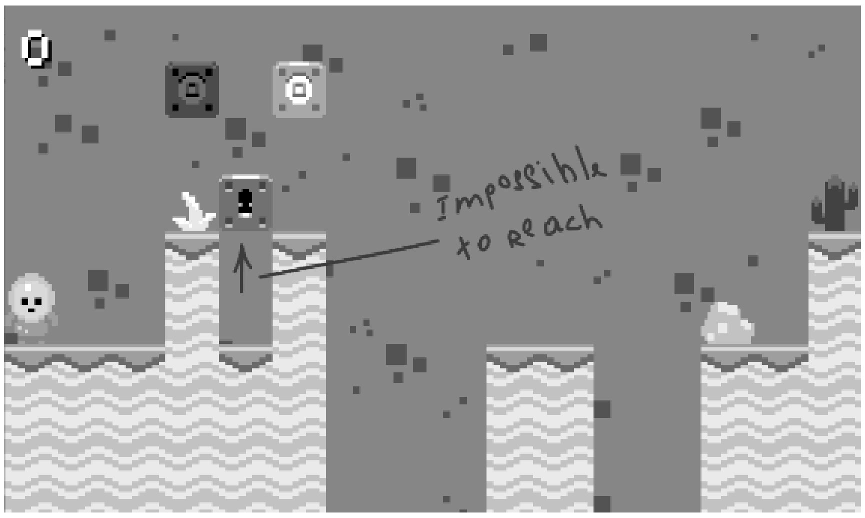

Fig 1.5 Issues with procedural generation

After getting acquainted with some of the details of the implementation of the main functions of the Mario project, it should be noted that this project is made exclusively for educational purposes. All sprites and art for the game were taken from following resource and are protected under Creative Commons License [7].

Conclusions and offers. The article reviews and analyzes the most popular game engines. Their positive and negative sides are considered. The practical experience of their application is described in detail, which will allow the developer to find an engine that suits his goals.

The choice of the cross-platform LOVE2D engine for creating Super Mario Bros as a simple, free, open-source engine is reasoned. The positive aspects are highlighted, the obvious disadvantages are indicated. The choice of the Lua programming language as a powerful, efficient, easy-to-learn scripting language is justified.

The article contains algorithm, code snippets, shows drawing functions, provides an opportunity to see how controls for a character are created. An attempt was made to create their own original condition for the behavior of characters in a certain way, that is, to create their artificial intelligence.

The authors created the Player class, which contains functions for checking collisions, and also implemented various states for the player to jump / fall / walk and idle.

Identification of "gaps" has also been implemented so that virtual enemies do not fall into them while chasing main character.

Also of note is the reason that makes this project a little more advanced than the original game: it's a procedural level generation. Artificial intelligence of enemies are the main differences from existing approaches used to create previous Super Mario projects.

The article demonstrates the ease of use of the basics LOVE2D using an example developed by the authors of the Mario project.

The obtained results made it possible to say that the work done is a new step forward in comparison with previous developments of algorithms for this 2D game.

The project presented in the paper is made exclusively for educational purposes.

References

1. Question and answer site for professional and enthusiast programmers. https://stackoverflow.com/
2. GitHub - code hosting platform for version control and collaboration. https://github.com/
3. Helper Utilities for a Multitude of Problems – Hump. https://hump.readthedocs.io/en/latest/
4. Class.lua for making Love2d object oriented. https://github.com/jonstoler/class.lua
5. Simple resolution-handling library. https://github.com/Ulydev/push
6. A collection of useful micro-modules for Lua. https://github.com/tst2005fork/lua-knife
7. A simple 2D pixel art style art for the game. https://opengameart.org/content/kenney-16x16.

к.т.н., доцент **Трегубова І.А.**
pisma-irine@ukr.net, ORCID 0000-0003-2030-7678,
itnek0pi@gmail.com, студент **Григоращенко В.О.,**

Державний університет інтелектуальних технологій і зв’язку,
м. Одеса

ВИКОРИСТАННЯ КРОС-ПЛАТФОРМНОГО РУШІЯ LOVE2D ДЛЯ РЕНДЕРИНГА ТА РОЗРОБКИ ШТУЧНОГО ІНТЕЛЕКТУ

Останнім часом зростає ринок консолей і мобільних ігор, а тому знайти ігровий рушій, що відповідає вибагливим вимогам користувачів – це не проста задача. Технологічні платформи стали очевидними фаворитами багатьох розробників. Проте, ринок мінливий, а тому питання вибору ігрового рушія своєї актуальності найближчим часом не втратить і є першим, основним, актуальним та вагомим предметом вибору в данній роботі. Актуальністю роботи було написання власного оригінального алгоритму для 2D комп’ютерної гри, який відрізняється від попередніх версій своєю унікальністю: новою процедурною генерацією, удосконаленим штучним інтелектом персонажів, оригінальною механікою гри, що викликало необхідність створення ключа для розблокування можливостей проходження рівнів.

Для роботи були обґрунтовано обрані крос-платформений рушій LOVE2D з відкритим кодом та мова програмування Lua, як потужна, ефективна та легка для засвоєння мова. При розробці алгоритму гри був використаний простий текст. Значна кількість питань було вирішено унікальним шляхом з нуля під час розробки. Створені орігінальні позиції механіки гри такі, як розблокування кінця рівня, нескінченність гри та орієнтація на набір максимального “Score”, збільшення величини кожного наступного рівня, в порівнянні з попереднім, удосконалення штучного інтелекту персонажів.Все це є основними відмінностями від вже існуючих підходів, які використовувались для створення попередніх проектів Super Mario Bros. Причина, яка робить цей проект більш просунутим і трохи більш випадковим, ніж оригінальна гра: це процедурна генерація рівнів. Оскільки гра нескінченна, дуже важливо, щоб рівні були випадковими і різними.

Отримані результати дали можливість говорити про те, що виконана робота являє собою новий крок вперед в порівнянні з попередніми розробками алгоритмів до даної 2D гри. Розроблені авторами статті оригінальний алгоритм та код для 2D комп’ютерної гри, з використанням можливостей сучасних інформаційних технологій, можуть бути корисними не тільки для створення мобільних ігор, а й при вирішенні задач віртуальної реальності, допоміжної реальності, в можливостях телепредставлення, при створенні візуалізації, ефектів галограмми, які

будуть реалізовуватися в мережах п’ятого покоління. Представлений у роботі проект зроблений виключно для освітніх цілей з метою впровадження в навчальний процес.

Ключові слова: крос-платформний рушій; мова програмування; процедурна генерація; алгоритм; код програмування; штучний інтелект