jLegends – online game to train programming skills

Konstantinos Tsalikidis, and George Pavlidis
tsalikidhs@gmail.com, gpavlid@gmail.com

***Abstract* - Gamification and in particular game-based learning is significantly gaining ground during the latest decades. It expresses a different approach to education that is mixing education with gaming, aiming to enhance the learning experience with game mechanics and rules and to provide stronger motivations for lifelong learning. The benefits of learning while playing have been illustrated by many works to this day. This work presents such a game-based approach that has been adopted and used in the development of an online multiplayer platform game, with a purpose to teach or train programming with JavaScript. In effect it is like what is usually called a *serious game*, or a game with a purpose. The game, *jLegends*, is online and available for everyone to train and test knowledge on programming and logic, within a role-playing gaming approach. jLegends is built with source-code scalability in mind, in order to be expandable or even become open-sourced in the future.**

Index Terms – game-based learning, gamification, educational games, serious games, computer science, programming, education, training, lifelong learning.

INTRODUCTION

The concept of educational entertainment (also referred to by the neologism *edutainment*) has a long history and has been used by media in many different forms such as audio, video, film, television, games and radio. Its main purpose is to attract and maintain an audience, while intentionally incorporating educational content. The process of learning is a very complex task which can be very imposing on the students since it requires a lot of effort. They need a lot of motivation to cope with it. In view of this, it is within the benefit of education to create educational software that can be more interesting and stimulating for students. Currently, there is a fast growing area of computer technology and computer games industry that is becoming appealing to youth, which aims to transform the difficult process of learning to become more amusing. The use of computers in education and training dates back to the early 1940s, when researchers developed the first flight simulators, which used analog computers to generate simulated onboard instrument data. The arrival of the personal computer, with the Altair 8800 in 1975, changed the field of software in general, with specific implications for educational software. While users prior to 1975 were dependent upon university or government owned mainframe computers with timesharing, after this shift users could create and use software (Fig. I) for computers at homes and in schools. Major developments in educational software in the early and mid-1990s were made possible by advances in computer hardware. Multimedia graphics and sound were increasingly used in educational applications. Optical media (such as CD-ROMs) became the preferred method for content delivery with several digital encyclopaedias being released as multimedia application CD-ROMs. With the advent of the Web in the second half of the 1990s, new methods of educational software delivery appeared. According to Richard Van Eck, there are three main approaches to creating software that stimulates cognitive growth in the gamer: building games from scratch by educators and programmers, integrating commercial off-the-shelf software (COTS) [1], and creating games from scratch by the students [2].

Game-based learning is a special case of gamification. Gamification is a process that targets the engagement and involvement of people in problem solving and development, in various environments and in a pleasing manner. Games can be used as an educational environment in order to learn about a specific subject or even develop a skill while playing. The use of games is based on the motivation aspect that games involve, which encourages curiosity and creates the impression of controlling the learning process. It has been shown that game-based learning can be combined with similar learning methodologies as Collaborative-based Learning, Problem-based Learning and Project-based Learning [3].

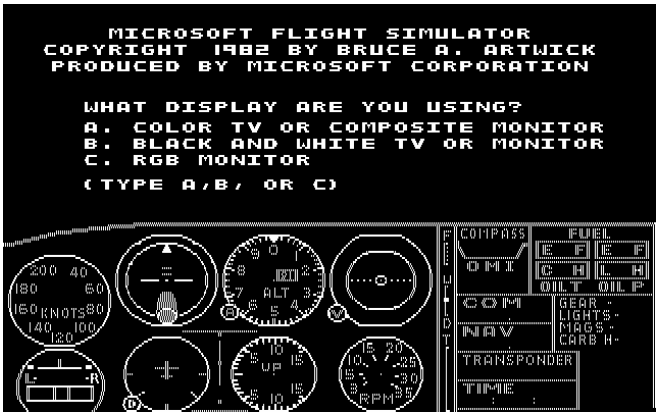

FIGURE I
THE 1982 PERSONAL COMPUTER FLIGHT SIMULATOR

Games often have a fantasy element that engages the player in a learning activity through a narrative or a storyline. Game-based learning is an expansive category, ranging from paper-and-pencil games from complex massively multiplayer online (MMO) and role-playing games (altogether MMORPG). The use of collaborative game-based role-play for learning provides an opportunity for learners to apply

acquired knowledge and to experiment and get feedback in the form of consequences or rewards, thus getting the experiences in a safe virtual-world. Real-world challenges are easier faced within a game containing effective, interactive experiences that actively engage people in the learning process. In a successful game-based learning environment, choosing actions, experiencing consequences, and working toward goals allows players to make mistakes through experimentation in a risk-free environment. Games have rules and structure and goals that inspire motivation [4].

Role-playing has a long history in Western culture from children's games through the theater and as a training method. However, fantasy role-playing as a commercial product was developed in the 1970s and made it to market as the well known *Dungeons and Dragons* (D&D, 1974) by Gary Gygax and Dave Anderson. Role-playing games (RPGs) are played in a wide variety of formats ranging from discussing character interaction in tabletop form to physically acting out characters in Live Action RPG (LARP) [5]. There is also a great variety of systems of rules and game settings. RPGs that emphasize plot and character interaction over game mechanics and combat are sometimes called storytelling-games. These types of games tend to minimize or altogether eliminate the use of dice or other randomizing elements. In 1978 the first *Multi User Dungeon* (MUD), a text based virtual reality (Fig. II) role-playing game was designed by Roy Trubshaw and Richard Bartle. It took almost twenty more years before the first three-dimensional Massive Multiplayer Online Role-Playing Game (MMORPG), *Meridian 59* (Fig. III) (1996) was released.

In general, in computers, *game development* is the process of creating a video game. Development is carried out by a game developer, which may be just one person or a multinational company. Traditional commercial PC and console games may take several years to develop. On the other hand *indie games* [6] can take less time and can be produced cheaply by individuals or a group of developers. The indie game industry has seen a rise in recent years with the mobile game market. The process of creating a video game starts from an idea or concept. Often the idea is based on a modification of an existing game concept. The game idea may fall within one or several genres as designers often experiment with different combinations of genres. Game development history begins with the development of the first video games. The first games created had little entertainment value as their focus was separate from the user experience. In fact these games required mainframe computers to play. In 1952 Alexander S.Douglas created *OXO* (Fig. IV-left), the first game to use a digital display. In 1958 Willy Higinbotham created a game called *Tennis for two* that displayed it's output on an oscilloscope [7],[8]. The success of "Space Invaders", an arcade shooter game by Taito started the beginning of the golden age of arcade video games and inspired many manufacturers to enter the market (Fig. IV-right). At the same time personal computers appeared, driving individual programmers and hobbyists to develop games on their own.

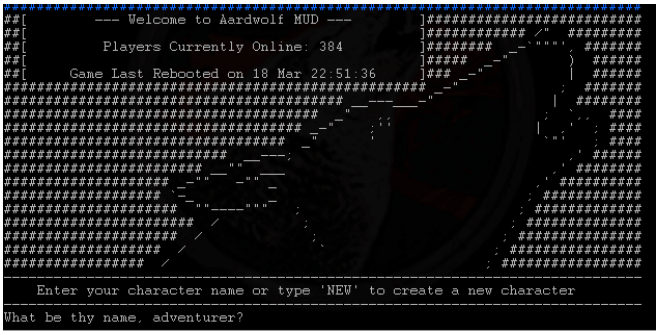

FIGURE II
USER INTERFACE OF A MUD USING A TEXT-BASED VIRTUAL REALITY

FIGURE III
SCREENSHOT FROM THE FIRST 3D MMORPG

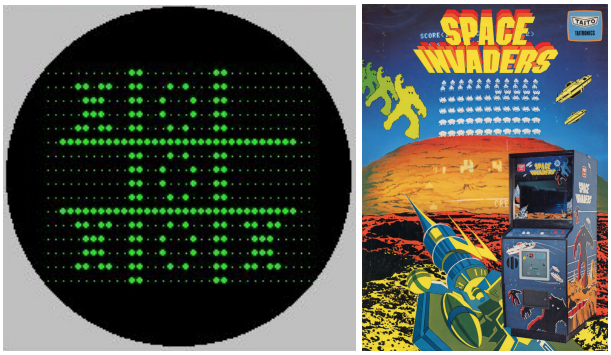

FIGURE IV
A RECREATION OF THE OXO DISPLAY FROM A SOFTWARE EMULATOR (SOURCE: HTTPS://GOO.GL/VF4PKT)

ON LEARNING HOW TO CODE

One does not need to be an engineer to program a computer. In fact, programming is like learning another language, which just happens to be easier the younger the person is. There are several online tools and platforms to start learning computer programming:

- *Code Academy*[1] is a famous website to teach code interactively, thanks to its helpful interface and well-structured courses.

[1] https://www.codecademy.com

- *Code Avengers*[2] offer courses designed to entertain and at the end of each lesson there is a mini game to release the accumulated stress and keep learners going for longer.
- Many more, like *Code School*, *Treehouse*, *CodeHS* and *Scratch*.

Using those tools anyone can start to learn programming interactively, easy, step by step and have, at the same time, some sort of fun.

Games can be really fun and exciting, and programming games can be a very interesting process, even if it is not to make a career out of it. Playing games using code seems to be an innovative, relatively new. concept. There are already several games online (there are some examples reported in the following paragraphs), played by using only code, with most of them targeting kids. Seymour Papert, a researcher at MIT, believed that learning programming could be much easier if there was a fun programming environment to play around in. This was so successful in the case studies he observed that he was forced to develop an explanation. He called his theory of learning *constructionism* [9] (because the students can construct their own knowledge by experiment). This spawned a genre of software toys designed to teach programming [10]. Some of the most popular coding-gaming initiatives are listed below:

- *Code.org*[3] is a non-profit organization and unanimous website that aims to encourage people, particularly school students in the United States, to learn programming by playing games
- *CodeCombat.org*[4] was largely an accident. After trying to learn how to code on his own, George Saines realized that the best way to motivate himself to code was to make it fun. Fast-forward. less than a year later and CodeCombat has become one of the fastest-growing companies in the *Learn To Code "Movement"*.
- *Code Spells*[5] started as the PhD research of Sarah Esper & Stephen Foster at UC San Diego to teach kids coding [11]. It is already being developed into something more than a research project, to make an immersive, visually-appealing video game that kids & adults will want to explore for hours.
- *Codehunt*[6]. Code Hunt is a an educational coding game in which the player has to discover missing code fragments [12].
- *Code Warriors*[7]. Code Warriors is a 3D action game developed by Kuato Studios and includes robots, JavaScript and battles, as the game guides the player from a beginner to advanced coder level [6].
- Other popular coding games or gamified coding environments are *Kodable*[8], *Codemancer*[9], *Scratch*[10] and *Tynker*[11].

JLEGENDS: AN RPG ABOUT CODING

The process of creating a video game starts from an idea or concept. *In this work the idea was to make an online role-playing game that would be played by writing code and code only (and no other means of keyboard or mouse interaction with the gaming environment), in order to train JavaScript programming skills*. Within this concept, if the game were to be used *by a single person then it would require a minimum general knowledge of the concepts of programming*; if the game were to be used *in a classroom-setting then a tutor familiar with the concepts of programming and with some experience in JavaScript would be adviced*. In the latter use-case, the students could use the game to learn an entirely new language under the guidance of their tutor in an engaging way. The game *at its full potential could include advanced programming scenarios*, as will later be described, in order to change even the very principles upon which the virtual world is built and the mechanics that drive it.

Generally, the creation of a role-playing game (RPG) starts with designing the story characters, a set of rules and settings, and, of course, the game mechanics, which constitute the most important part regarding the user experience. Then the graphics come into play, in order to visualize the environment and the characters and to provide the stage for the action. Last, but not least, the choice of a gaming platform to target (at least in the beginning), and the rest of the technologies (libraries/platforms) that will be used to support the development process. The characters in such application scenarios are usually distinguished by two basic characteristics; the *race* and the *class*. The race represents the *species* to which the character belongs. The class assigns *abilities* and *attributes* to the character. For example a *Warrior* fights in close range using his sword, however a *Ranger* attacks from a long range using a bow. A *Cleric* or *Priest* is specialized in healing and all sorts of supportive magical abilities. In some systems the player is free to choose a specific path about his/her character usually by assigning points to certain skills. At the end characters end up being a mix of different “ingredients” and changed by player's choices.

In jLegends (JavaScript Legends), the game created for the purposes of this work, four (4) types of character races have been selected: *Humans, Elves, Orcs, Undead*, and six (6) types of character classes: *Warrior, Tank, Ranger, Mage, Priest*. Every character (even non-playing characters-NPCs) is also defined by the following attributes: *Strength, Agility, Vitality, Energy*. Additionally, each character is described by the following properties:

- *ad – Attack Damage*, which empowers physical attacks
- *ap – Attack Power*, which empowers magical spells
- *armor – Armor* that protects you from physical attacks

[2] https://www.codeavengers.com
[3] https://code.org
[4] https://codecombat.com
[5] http://codespells.org
[6] https://www.codehunt.com
[7] http://www.kuatostudios.com/games/hakitzu-elite
[8] https://www.kodable.com
[9] http://codemancergame.com
[10] https://scratch.mit.edu
[11] https://www.tynker.com

- *life – The amount of life hitpoints*, which are being consumed by enemy attacks
- *mana – The amount of mana points*, which are being consumed by using spells

These properties are calculated using pre-defined rules and according to percentages declared in the settings of the game (Table I). These character stats are being used by the *combat system* in order to simulate a fight in a simple and straightforward manner.

TABLE I
HOW CHARACTER STATS ARE CALCULATED

Character	ad	ap	armor	life	mana
Warrior	Sx1.5	Ex0	Sx0.25+Ax0.75	Vx10	-
Tank	Sx0.75	Ex0	Sx1.5+Ax1	Vx20	-
Ranger	Ax1.5+Sx0.25	Ex1.5	Sx0.25+Ax0.25	Vx7.5	100+(Ex1.5)
Mage	Sx0.75	Ex4	Sx0.25+Ax0.25	Vx5	100+(Ex3)
Priest	Sx0.75	Ex2	Sx0.25+Ax0.25	Vx5	100+(Ex4)

** S ← Strength, A ← Agility, V ← Vitality, E ← Energy*

The rules and settings of the game, play a crucial part in the creation of an RPG. The rules are what makes the game interesting and keeps the players under excitement while playing. The settings are dynamic variables, which tend to change over time in order to achieve a balance between the different paths and choices that are available to the players. Rules apply to each player separately. Every player can create an infinite number of characters. Every character is described by its race, class, stats, power level and a name. Players can only create new characters of Humans and Elves, while Orcs and Undeads are used as non-playable characters (NPCs) controlled by the artificial intelligence (AI) system of the game. Players can choose between three (3) different game modes:

- *Single Player* - Playing alone versus the game AI. Characters gain experience points by killing mobs, which eventually lead into leveling up. This is a mode either for learning the game or for experimenting.
- *Multiplayer* - Playing with others versus the game AI. Characters gain experience points by killing mobs and level up through collaboration.
- *Player versus Player (PVP)* - Playing versus other players, as a team or individually. In this mode players are able to form a team and play against others. It can be really challenging to play by using tactics against other players. PVP mode uses a rating system to produce rankings for the players.

Every character is able to perform the following standard actions:

- Horizontal movement (left/right)
- Attack other characters
- Cast a spell to any other character
- Use a skill
- Rest to recover

The spells and skills vary according to the class of character.

jLegends is a turn-based game. All characters are able to perform no more than two (2) actions per turn. When playing against the game AI, in each turn the player (or players) plays first. Every time a player dies, loses a short amount of experience but re-spawns after three rounds. At the end of each turn all characters gain a small amount of *mana* and *hp*. The battleground consists of twenty five sectors where characters can move, stand and attack or cast spells. All characters gain two extra points for each power level in order for the players to build their character's main stats (strength, agility, vitality, energy).

jLegends is a browser-based game that can be played using any modern browser even in mobile devices regardless of the operating system. The server-side is written in JavaScript as a *Node.js application*. The client is written in HTML, CSS and JavaScript using several modern libraries and modules, as will be described in the following paragraphs.

jLegends is played exclusively by writing code[12], so the game itself is responsible for running the code submitted by the player, produce an output and translate this output into real game actions. As the output is being translated, the engine must ensure that the actions do not violate the game rules or lead to an undefined result. The game engine of jLegends consists of several classes such as *Player*, *NPC*, *Actions*, *Skills*. All these classes, written in JavaScript, compose the game engine by cooperating. Fig. V shows the relations between several classes in order to compose a Player class.

The *Stats* class represents the ability of an object to be described by specific properties such as attack-damage, ability-power, armor, etc. *Warrior, Ranger, Mage, Priest* classes represent the game characters classes. Every Player class inherits from one of these classes. The *Actions* class represents the ability of the Player to perform any kind of action: basic or special moves, simply moving around to using a spell. The *Orientation* class represents the ability of the Player to "know" its position, including the distance from any given point in the battlefield, and also distances from enemies and allies. *BuffMechanics* represents the ability of the Player to receive buffs. The *NPC* class is almost identical to the Player but with a few differences and controlled by the game AI.

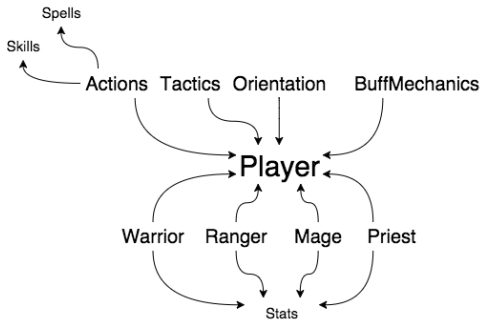

FIGURE V
PLAYER CLASS IN JLEGENDS

[12] An interactive tutorial on how to use and exploit the basic game functionalities can be accessed at http://jlegends.io/tutorial.

II. Technical aspects of jLegends

For the development of jLegends *Node.js (NodeJS)* has been employed. NodeJS is an open-source runtime environment for developing server-side applications. Applications are written in JavaScript or any other language that compiles to JavaScript, such as TypeScript, Dart, etc. NodeJS was created by Ryan Dahl in 2009 and the biggest difference from PHP and Apache is that Apache is thread and process based (i.e each request is handled by a separate thread or process), which means that while it is processing input/output the entire thread is blocked. NodeJS has asynchronous, event driven input/output. Every NodeJS instance runs in a single thread and due to its asynchronous nature, it can handle far more concurrent requests compared to Apache.

Another core technology used to develop the server is *web sockets*, the bi-directional, full-duplex, persistent connections from a web browser to a server. Once a web socket connection is established the connection stays open until the client or server decides to close this connection. With this open connection, the client or server can send a message at any given time to the other. This makes web programming entirely event driven, not just user initiated. Web sockets provide a persistent connection between a client and server that both parties can use to start sending data at any time.

Relational databases are the most common database management systems. They include databases like SQL Server[13], Oracle Database[14] and MySQL[15]. A relational database management systems (RDMS) offers much better performance for managing data over desktop database programs. Today, the most innovative structures for storing data are NoSQL [13] and object-oriented databases [14]. These do not follow the table/row/column approach of RDBMS. In jLegends *MongoDB*[16] was used, which is a NoSQL database. Mongo database is open-source, developed by MongoDB Inc[17]. MongoDB stores data in JSON-like documents that can vary in structure. Related information is stored together for fast query access through the MongoDB query language.

The client was written exclusively using HTML5 [15], CSS3 [16] and JavaScript [17]. One of the main technologies used to develop the client is *jQuery* [18]. jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and AJAX much simpler with an easy-to-use API that works across a multitude of browsers [18]. The client in it's core however is developed with *AngularJS* [19], an open-source framework maintained by Google and individual developers of the open-source community. AngularJS simplifies both the development and the testing of web applications (client-side) providing an elegant *Model-View-Controller (MVC) Framework* [20] along with many components commonly used on web applications. One of the main aspects of AngularJS that makes it extremely useful for jLegends also is the *data binding feature*. Data binding in AngularJS is the the automatic synchronization of data between the Model and View components. The View is a projection of the Model at all times. When the Model changes, the View reflects the same changes and vice versa [21].

Another core technology used for the client of jLegends is *Photon Storm Phaser (Phaser)*[18]. Phaser is an open-source HTML5 game framework designed to create browser games. If the device is capable then it uses *WebGL*[19] for rendering, but otherwise it seamlessly reverts to the *Canvas* object. Phaser has a built-in asset loader that can handle: Images, Sprite Sheets (fixed sized frames), Texture Atlases (including Texture Packer, JSON Hash, JSON Array, Flash CS6/CC, and Starling XML formats), Audio files, Data Files (XML, JSON, plain text), JavaScript files (so you can part-load your games or JS based resources), Tilemaps (CSV and Tiled map formats) and Bitmap Fonts.

IMPLEMENTATION

Currently, jLegends is in beta version so only single-player and multiplayer modes are available. Fig. VI presents a screenshot during gameplay in jLegends.

The interface consists of a *code editor*, that provides a textbox for the user to write code, the actual game stage and some tools and options. The code editor supports syntax highlighting for JavaScript, line numbering and a keyboard shortcut to save the code of the user (⌘+S/Ctrl+S). At its current version the interface does not support a multiple file system. To start playing jLegends it is enough to write 5 to 10 lines of code but as users go through the game they realise that they need more and more code to structure a more complicated logic. The game canvas aligns at the center of the window while keeping a fixed aspect ratio.

Tools and *options* are provided at the bottom right of the game interface. The *Play* button and an *Auto-play* switch are shown along with the *Methods* button and the *Quit Game* option. The *STATS* tab displays the character's stats and lets the user redesign the character.

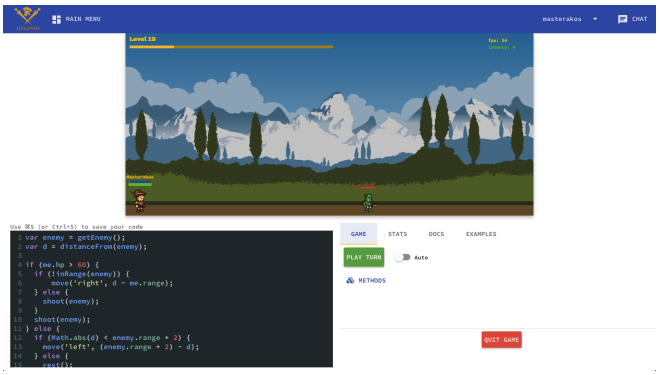

FIGURE VI
JLEGENDS GAMEPLAY SCREENSHOT

[13] Microsoft SQL Server, http://www.microsoft.com/en-us/server-cloud/products/sql-server/
[14] Oracle Database, https://www.oracle.com/database
[15] MySQL Database, https://www.mysql.com/
[16] https://www.mongodb.com
[17] https://www.mongodb.com/company
[18] Photon Storm Ltd., *Phaser* at http://phaser.io
[1919] Khronos Group, *WebGL* at https://www.khronos.org/webgl/

The *DOCS* tab contains everything the user has to know in order to play the game. All methods to start writing code are organized like a simple technical documentation. Although the DOCS represents a quite easy way to guide through, there is also another mini window to remind of all methods, like a paper sticker, enabled by the *Methods* button.

Another core component is the *chat window*. It folds and unfolds by the user. The chat takes place to a public channel between all users currently playing jLegends. In case of joining a public game, the chat is limited to users playing in the same game as allies.

It should be stressed that the most complex part of the interface implementation was handling the user code and executing it, in order to produce game actions limited by the rules of the game.

I. jLegends game mechanics

The overall architecture consists of the client and the server exchanging data according to the user's input and the game rules. This is exactly how an ordinary online game works with the only difference of having programming code as user input. Fig. VII summarizes the overall functionality through a typical sequence diagram. The *User* writes a piece of code to play, whereas the *Client* executes this code and produces an output, which is sent directly to the *Server* in order to validate and perform game-actions. After-effects are sent back to the *Client* in order to be displayed for the *User*. This architecture allowed to isolate the code-execution process to the *Client*. This way the *Server* has less and a lot safer job to perform. Unsafe code and CPU resources consumption take place exclusively on the *Client* bypassing a huge amount of security code that would have been needed otherwise.

In order for a user to play a turn, the user simply clicks the *Play Turn* and if the code is valid, the next turn is being played automatically. However, many more things happen before anything is being shown in the game scene. What doesn't happen and should be noted is that the client is not sending any code to the server. The client instead executes the code and produces an output (JavaScript object), which describes game actions. These actions will later be sent to the server for validation before they take place to the actual game scene. Code execution is handled by the *Sandbox* class. The Sandbox class contains all the available methods according to the rules and settings of the game.

An experienced web developer would easily be able to edit this class from client's source-code using a modern browser and literally change what these methods do. Then, a pre-defined method such as *move* could implement something entirely new instead of just moving the character (as pre-defined). This would lead to a completely *hackable game*, resulting the elimination of any competition between competitive players of the game. Of course this cannot really happen, at least at the current version of the game, because as it has been said, the output will be validated later by the server and only rule-based valid moves will be executed.

After the client has executed user's code and produced an output, the output is being sent to the server through a web socket connection. Before any further execution, the server tries to make out the number of actions the user requested in order to make sure that the number of actions is less than or equal to the permitted actions per turn (according to the game rules and settings). This is a security measure against any possible hacking action trying to send thousands or millions of actions, to overwhelm the server. Everything else is handled by the *Engine* class, which also controls the NPCs.

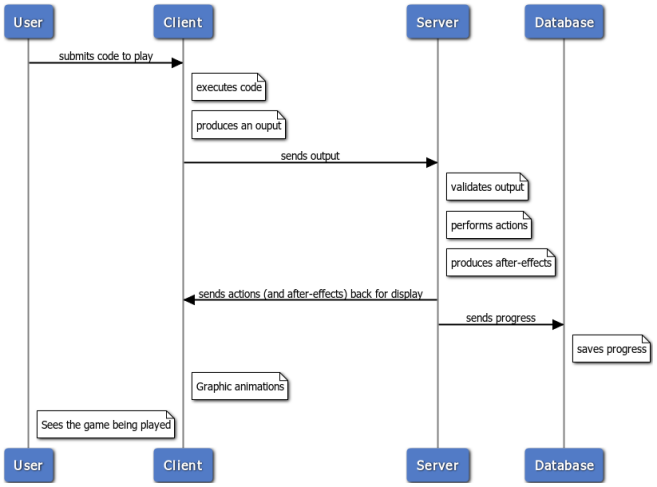

FIGURE VII
SEQUENCE DIAGRAM OF PLAYING A USER TURN

After each turn by all players, NPCs respond (in player versus AI). These NPCs are being played by the game AI written within the *Engine* class. At the current game version the AI is a simplified system with minimal intelligence. The current AI algorithm requires three (3) parameters:

- *start-point*, the *x*-coordinate of the spawned NPC. This parameter helps to limit the random behavior of an NPC, or simply, to know where it started and how far can it go
- *patrol-range*, a number that defines the range the NPC can patrol. The NPC will not exceed this range without a reason
- *reset-agro-range*, a number that describes the range at which the NPC will reset its aggression (*agro*) towards the current target. If the NPC has gone too far from its start-point (in order to attack this target) then it will reset its *agro* and stop pursuing the target.

Depending on these parameters the NPC may have a passive-normal role in the game or a highly aggressive stance. The algorithm uses three different methods repeatedly in order to produce two (2) actions. As it's shown in Fig. VIII, the algorithm starts by calculating the *agro* given the current circumstances in the environment. *Agro* is calculated by how close is the enemy or how strong its attacks are expected to be to qualify as a threat and increase the *agro* for this character. If there is enough *agro* for any of the enemy characters, then the AI decides to attack this character. However, if there is no *agro* at all and everything seems clear, then the AI will move the NPC randomly inside a pre-defined patrol area (defined by the patrol-range parameter mentioned before) and re-calculate the *agro* accordingly.

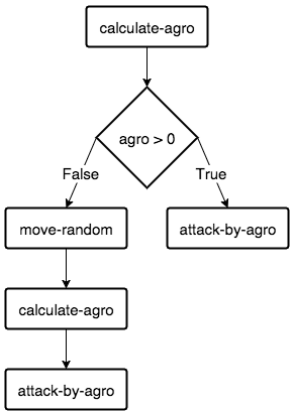

FIGURE VIII
GAME AI FLOWCHART

```
var enemy = getEnemy();
moveTo(enemy);
attack(enemy);
```

FIGURE IX
SIMPLEST PHYSICAL ATTACK EXAMPLE

```
var enemy = getEnemy();
cast('ignite', enemy);
```

FIGURE X
SIMPLEST RANGED ATTACK EXAMPLE (FOR MAGE CHARACTER)

```
var enemy = getEnemy();
if (me.mp < me.mana*.4) {
  rest();
} else {
  cast('ignite', enemy);
}
```

FIGURE XI
ADVANCED RANGED ATTACK EXAMPLE

Fig. IX presents the simplest example of a code to deal with the enemy in the game. In this piece of code, the user uses a typical physical attack on the enemy to inflict damage. The way it works is based on the exploitation of a classical object oriented approach, where the *getEnemy()* function is responsible for reading all the data regarding the *enemy class object* and storing them to the *enemy* variable. Then the second command, executes a movement, which sends the user avatar towards the enemy and in a physical attack range. The last (third) command executes a physical attack, when the user character is able to perform such an attach. In a similar manner, the piece of code in Fig. X executes a ranged attack, when the character supports such an attack. Again, the first thing to do is to read the enemy data using the *getEnemy*(). A more complicated and advanced tactic is implemented in the piece of code depicted in Fig. XI. In this tactic, the user checks its current *mana* status (*mp* variable) and if there is not enough mana (energy) to cast a ranged attack, the player just *rests* to recover faster. In case there is enough energy the player casts the ranged attack spell *ignite* that is related to a *mage* character in this case.

DISCUSSION

jLegends has been designed and developed as a simple and easy-to-use MORPG to train programming skills. At its current implementation it employs a stable and highly functional framework as a foundation, and is based on a simple platform gaming interface and a basic-level game AI[20].

The single player mode is definitely a mode of a player against NPCs. In multiplayer mode the main scenario includes multiple users against NPCs. At the current version the system waits for all users to hit "Play" to complete a "turn". We are currently exploring approaches like:

- the players to be able to see who has already completed (pressed "Play")
- to use a global clock and to keep the game within time limits: any player would have only, say, 1 minute to play, or maybe 1 minute after the 1st player has completed

The PVP mode is also in beta and under development (and not published yet), where the same ideas are being considered.

Considering the gaming scenario, initially, a player should emphasize on creating a hero and forming the character's qualities. The player should strive to increase the character's level and to appropriately share the gained points among the character attributes. This should be carefully balanced and in the framework of the gaming experience the players want to have. For example, a warrior with 100 HP points (and low strength) has a huge amount of life and can take a lot of beating. On the other side, a warrior with 50 life and 50 strength can withstand some attacks and impose considerable damage when attacking, at the same time. The 100-HP player has excellent defense against NPCs but is practically "useless" in PVP, since in such a game the hero with the less HP is going to be the target of the attacks and not this hero (who would somehow be considered as a "shield").

Once the players form the characters of their heroes then the challenges are relevant to the "quality" of the code they use and their strategy. This approach is typical and common in many RPG games of this kind. It should be stressed that at the current version the game AI is very basic and supports just two types (warrior+ranger) of NPCs and only the level of the NPCs changes when the player advances a level. Ideally, *we imagine the player to pass through the following phases* (and we are currently working towards these scenarios):

- *Starts playing by experimenting* with code in order to get accustomed and to learn how to use the system using all the functions the system provides. Possibly the player changes the code frequently to check the outcome.
- Understands that the *auto-play feature* may be exploited to jump a few levels
- Then *new problems should arise* (new NPCs with new playing approaches) in order to make the player think about revising the code
- Ultimately, the player could use *multiple code files for various cases and strategy styles*

There are a number of points that still represent challenges for further development. For example, *a multiple*

[20] The game is online and available for testing at http://jlegends.io.

file system and tabs to navigate betweens files would be a much better solution, instead of the single file support at the current version. Additionally, having only 15 *lines of code visible in the display* is a bit uncomfortable for the user. Another point to explore for further development raises from the fact that the chat system does not *support a private messaging mechanism* to allow two users to communicate privately. In addition, there seems to be a drawback in the adopted *architecture logic*: as the *Client* executes *User* code and produces a number of actions, all of the sequence of actions is being executed regardless of any intermediate outcome (ex. could attack an already dead character). Last but not least, a *better AI algorithm* could be a great asset in the game as it would offer more challenging gameplay and engagement in the game.

CONCLUSIONS

As more and more educational approaches pave a way towards a really gamified education there is still a lack in the gamified training in programming. To help fill this gap, an MORPG, jLegends, has been designed and developed based on the platform gaming paradigm, with a purpose to teach or train on programming in JavaScript. jLegends was built upon stable and innovative we-based technologies and is available for everyone to train and test programming knowledge, styles and logic, within a well-defined role-playing gaming approach. Currently we are exploring ways to upgrade the game engine, to provide more complex AI, to support multiple code files per user, and add more player-to-player and player-to-AI playing options. In addition, we are scheduling real-life evaluations in classroom setups.

ACKNOWLEDGMENT

This work is in partial fulfilment of the requirements for the MSc degree in "Innovation in Technology and Entrepreneurship", Dept of Electrical Engineering, Eastern Macedonia and Thrace Institute of Technology.

REFERENCES

[1] D. McKinney, *Impact of Commercial Off-The-Shelf (COTS) Software and Technology on Systems Engineering*, Presentation to INCOSE Chapters, August 2001.

[2] R. Van Eck, *Digital game-based learning: It's not just the digital natives who are restless*, Educase Review, 41,2, 1-16, 2006.

[3] J. Hamari, J. Koivisto, H. Sarsa, *Does Gamification Work? – A Literature Review of Empirical Studies on gamification*, In proceedings of the 47th Hawaii International Conference on System Sciences, Hawaii, USA, January 6-9, 2014.

[4] J.D. Shearer, *Development of a Digital Game-based Learning Best Practices Checklist*, Electronic Thesis or Dissertation. Bowling Green State University, 2011.

[5] K. Salen, E. Zimmerman, *Rules of Play: Game Design Fundamentals*, The MIT Press, 2003, ISBN 0-262-24045-9.

[6] Kuatostudios, *Hakitzu Elite*, online at: http://www.kuatostudios.com/games/hakitzu-elite, latest access: Nov, 2015

[7] M.E. Moore, J. Novak, *Game Industry Career Guide*. Delmar: Cengage Learning, 2010, ISBN 978-1-4283-7647-2.

[7] J. Anderson, *Who Really Invented The Video Game?*, Atari Magazines, Retrieved November 27, 2006.

[9] S. Papert, *Mindstorms: Children, Computers and Powerful Ideas*, Prentice Hall / Harvester Wheatsheaf, New edition 1 Sept. 1982, ISBN: 978-0710804723.

[10] R. Lockhart, *Games that teach programming: a brief overview*, web resource: http://goo.gl/6KyCEY, latest access: Nov, 2015

[11] Codespells, *Codespells*, online at: http://codespells.org/, latest access: Nov, 2015

[12] CodeHunt, *About CodeHunt*, online at: https://www.codehunt.com/about.aspx, latest access: Nov, 2015

[13] MongoDB, *NoSQL explained*, online at: https://www.mongodb.com/nosql-explained, latest access: Nov, 2015.

[14] The Computer Technology Documentation Project, *Object oriented databases*, online at: http://www.comptechdoc.org/independent/database/basicdb/dataobject.html, latest access: Nov, 2015.

[15] World Wide Web Consortium (W3C), *HTML*, online at: http://www.w3.org/html/, latest access: Nov, 2015.

[16] World Wide Web Consortium (W3C), *CSS*, online at: http://www.w3.org/Style/CSS/, latest access: Nov, 2015.

[17] Press release announcing JavaScript, *Netscape and Sun announce JavaScript*, online at: https://goo.gl/HsLWLk, PR Newswire, December 4, 1995.

[18] jQuery, *What is jQuery*, online at: https://jquery.com/, latest access Nov, 2015.

[19] Google, *AngularJS*, online at: https://www.angularjs.org, latest access Nov, 2015.

[20] Cunningham & Cunningham, Inc., *Model-View-Controller*, online at: http://c2.com/cgi/wiki?ModelViewController, latest access: Nov, 2015

[21] K. Tsalikidis, *Two-way data binding demonstration using AngulatJS*, online at: http://jsfiddle.net/Tsalikidis/rfzqwgyb/4/ddle, latest access: Nov, 2015.

AUTHOR INFORMATION

Konstantinos Tsalikidis, MSc candidate, MSc in Innovation in Technology and Entrepreneurship, Department of Electrical Engineering, Kavala Institute of Technology, Kavala, Greece.

George Pavlidis, senior member IEEE, Research Director, Multimedia Research Group, 'Athena' Research Center, Xanthi, Greece.