PyDiophantus Maze Game: Play it to Learn Mathematics or Implement it to Learn Game Programming in Python

Dimitra Koupritzioti, Stelios Xinogalos

Abstract

Serious games or educational games have attracted the interest of instructors and researchers for several years. In the field of education, serious games are being used for motivating students, attracting their interest in cognitively demanding fields and making the teaching and learning process more fun. Moreover, learning through implementing games has been proposed as an effective alternative to the traditional instructor-centered approach to teaching programming. Towards this direction the study presented in this article aims to investigate whether it is feasible to learn programming concepts, as well as game programming concepts, through implementing a game in Python. More specifically, the study presented has the following goals: firstly, to review and comparatively analyze existing game engines and libraries that can be used by novice programmers for implementing simple games in Python; secondly, to investigate whether it is feasible to implement a simple but yet meaningful game that can be used as a prototype for learning programming concepts and game programming concepts in Python. In order to reach the second goal a serious game was implemented as a case study, using the free and open source Python library of pygame that based on the aforementioned comparative analysis is appropriate for novices. The PyDiophantus Maze game that was implemented can be used for teaching and learning game programming in Python, but also for learning mathematics. The article concludes with proposals for utilizing the game in mathematics and (game) programming education, as well as plans for further research.

Keywords: serious games, educational games, mathematics, programming, game programming, game library, Python

1. Introduction

Game-based learning is a term used to describe any initiative that mixes computer games (videogames) and education. This may entail anything between the introduction of commercial games in educational processes and the application of slightly interactive multimedia wrappers around traditional educational content (Moreno-Ger et al., 2008). Consequently, the design of *serious game*s or *educational games* is a broad subject that groups very different approaches and methodologies. Balancing fun/entertainment and education is an important concern in the design process. On one extreme, efforts may focus on translating the official educational content into a game-like environment, such as adding playability on top of preexisting learning material. On the other extreme, commercial games whose contents and models are rich and detailed can be used for educational purposes in ways that even their creators had not envisaged. A middle ground approach involves the use of games specifically designed for learning. Several challenges are inherent in this approach, such as the difficulty of designing a truly entertaining and engaging game with learning goals/tasks involved and the associated costs which may be substantial and hard to justify.

Given that game design is not an exact science and fun is a subjective concept, results may be mixed.

A *Science Learning Game* is a computer game whose purpose is to educate the player on a STEM (Science, Technology, Engineering or Mathematics) topic through gameplay. Such games can specifically support formal or informal STEM education, where alignment with the school curriculum may be a consideration. In another scenario, outside the education system, players may be engaged in "crowd science" where they help solve pieces of complex puzzles or problems that are of interest to scientists conducting research. Problems may be factored into pieces addressable through collective or crowdsourced multiplayer game play sessions (Yampolsky & Scacchi, 2016).

Besides the interest of instructors and researchers on using educational games in STEM education or education and training in general, there is also a great interest on teaching and learning (serious) game programming (Xinogalos, 2018) to and by students respectively. However, building video games is an effort-intensive and error-prone software-development task (Guana et al., 2015). So, *game engines* which offer reusable components that simplify common game related tasks like rendering, physics related computations and input are usually used to bring a game to reality and even more to teach game programming to novices. This frees developers from having to write code to handle these tasks from scratch so that they can focus on the details that make their game unique (Paul, Goon, & Bhattacharya, 2012).

This article presents a study investigating the potential of implementing a game in Python as a means of learning programming concepts, as well as game programming using Python. In order to support novices in implementing such a game, a review and comparative analysis of Python game libraries was carried out. Based on the results of this comparative analysis, pygame was used for implementing a serious game for teaching and learning mathematics as a case study. This game is analyzed in terms of the programming concepts/constructs or game programming concepts that can be taught to students either by presenting it as a case study or by asking students to implement it. The educational game that was implemented is a maze game called PyDiophantus and is targeted to STEM education. Although PyDiophantus was implemented as a case study it is fully functional and can be utilized for practicing on evaluating arithmetic expressions and priority of operators.

The rest of the article is organized as follows. In section 2 the methodology of the study is presented. Section 3 focuses on the issue of selecting a Python game engine or library that is appropriate for novices and presents a review of relevant work and a comparative analysis of six free and open source Python game engines and libraries. In section 4 the gameplay and the configuration capabilities of the serious game that was implemented as a case study are presented. Section 5 presents an overview of the (game) programming concepts and constructs utilized in the game and proposes various possibilities of utilizing it in (game) programming education. The article concludes with proposals for further research.

2. Methodology of the study

The study presented in this article has the following research questions:

> ***Research question 1 (RQ1):*** *What is the best free and open source game engine or library that can be used by novices for implementing simple games in Python?*
>
> ***Research question 2 (RQ2):*** *Is it possible to implement a simple but yet meaningful game that can be used as a prototype by novices for learning programming concepts/constructs and game programming concepts in Python?*

In order to investigate RQ1 a review of existing literature on reviewing, analyzing or comparatively analyzing game engines was carried out. Although several studies exist in the literature, we were only able to locate just a few references on Python game engines and libraries. However, this is not surprising since Python has gained a great interest as a programming language the last years. Based on the criteria used in the relevant work six free and open source game engines and libraries for Python were studied. A review and comparative analysis of these engines is presented.

Taking into account the results of the review and comparative analysis carried out in the context of RQ1, a game prototype was implemented for investigating RQ2. The game was analyzed in terms of the fundamental programming concepts/constructs, as well as the game programming concepts that it includes and as a consequence that could be learnt by novices through studying the game as a case study or implementing it with or without guidance from an instructor.

3. Review of Python game engines and libraries

3.1 Related work

Several papers comparing game engines have been published but they are typically not limited to a single scripting language but rather mostly focus on the type of game.

Vasudevamurt et al. (2015) define a comprehensive grouping, analysis and comparison framework for game engines, focusing on engines that can be used for the development of "*serious games*" which they define as games that have a non-entertainment goal and context. In their analysis, the authors consider more than 100 serious games and the engines they were built on and they separate engines in three groups based on their popularity, namely, actively used (very popular), somewhat used (not so popular) and next generation (recently released with cutting edge features).

Andrade (2015) conducts a survey about popular game engines and after presenting the main features of each engine presents a comparative table summarizing the game genre and type of graphics supported (2D/3D), the publishing target (where developed games can be played), the starting price for commercial usage, the scripting language and the supported platforms where developers can work. The overall conclusion is that there is no such thing as "the best game engine". The choice of a game engine should be based on the kind of game, the prior experience of the developer and how much effort they are willing and are able to put into its development and if required on learning a new platform. An additional consideration is

whether the developer is getting into the game industry for the long run or just trying to get acquainted with the technologies. In the first case, the time required to learn a complex engine with many features may be justified while in the second one, a simpler engine with fewer components may be just enough to provide a working prototype. Developers that have no prior experience in building games should also take the time to experiment with a couple of different options and get a feel of different editor interfaces and game abstractions. Additional factors that need to be considered before reaching a decision are the size and activity of the game engine's community, as a larger and more vibrant community means that there will be more support and the breadth, depth and quality of the engine's documentation and samples.

Trenholme and Smith (2008) present a survey on game engines for developing *first-person virtual environments*. The parameters they summarize in their final table are the availability of an integrated level editor, whether the game source code is available, the software development kit (SDK) bundled with the engine, its documentation and various other parameters such as the size and activity of the community, whether certain features require paid licenses and others. The authors note that in many cases it would be hard to imagine an application for which one game engine would be suited and others not. All the engines they consider provide some source code to users, typically partial that can be altered and recompiled to affect the gameplay. The existence of a strong and active online community supporting the engine, particularly with official developer support, is considered an important plus as is the minimum required specification for off-the-shelf hardware.

Cowan and Kapralos (2017) also focus on engines and frameworks used for *serious games*. In order to come up with a list of engines for comparison, authors surveyed three databases of academic publications, namely Google Scholar, ACM Digital Library, and IEEE Xplore Digital Library for names of game engines mentioned in papers related to research on serious/educational games. The 20 most popular engines (in academic writings) were then searched on Google to determine their popularity. In a separate survey, authors attempted to discover which engines were most popular among serious game developers by considering publications where an actual serious game was created using a particular engine. By combining these two surveys authors ended up with a list of 10 game engines (with only one using Python as a scripting language – Cocos). The parameters they summarized in a table for the selected engines where the level editor, the support for scripting, the support for C++, networking, 3D graphics, shader effects, support for dynamic shadows, physics, artificial intelligence, licensing (free for non-commercial/ free for commercial), support for mobile devices and a web player.

Christopoulou and Xinogalos (2017) focus on comparing game engines for desktop and mobile devices (Android and iOS). The comparison framework was based on several detailed features of the engines. The two most popular engines, namely Unity and Unreal Engine 4 were then selected for a hands-on, empirical study where the engine's usability, the learning curve, the process of exporting for mobile devices, as well as the game's quality and application size were evaluated.

It is clear that the relevant work on game engines has not investigated engines and libraries based on Python, with just a few exceptions. However, this is not surprising since Python has gained a great interest as a programming language the last years.

3.2 Overview of Python game engines and libraries

In this section six free and open source game engines or libraries supporting Python are presented, as those can be tested and used in classrooms without additional costs. Notable engines that are commercial products, such as BigWorld (http://bigworldtech.com/) which is a middleware platform for the development of massively multiplayer online games and virtual worlds, and Shark 3D (https://spinor.com/), a software solution for creating and running interactive virtual 3D worlds are not presented.

pygame Engine

pygame (https://www.pygame.org) is a free and open source software library written in C and Python whose purpose is to facilitate the creation of multimedia applications and games. Pygame is built on top of the Simple DirectMedia Layer (http://www.libsdl.org/) which is a popular free and open source library that provides low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D. SDL & pygame are highly portable and run on any device and operating system. The initial release of pygame was back in 2000 so it already has a long history in programming terms.

Pygame is supported by a vibrant community of developers and users who contribute to an extensive documentation, tutorials and videos with easy to follow examples and sample code. This makes pygame ideal for a classroom environment or a programmer that dabbles with game creation for the first time.

It must also be noted that other popular engines or libraries are built on top of pygame, such as Ren'Py, a popular software tool for virtual storytelling.

Pyglet Engine

Pyglet (https://bitbucket.org/pyglet/pyglet/wiki/Home) is a cross-platform windowing and multimedia library for Python intended for games and multimedia applications. Pyglet supports windowing and event handling, OpenGL graphics, music and sounds and runs on Windows, OS X and Linux. Pyglet only requires the installation of Python and does not have any other external dependencies. It is distributed under the BSD license.

Pyglet allows the developer to utilize multiple games windows and multiple monitors/ desktops. Practically any format of images, audio and video files is supported using the FFmpeg encoding algorithm while there is built-in support for popular formats such as wav, png, bmp, and others. Although Pyglet is written in pure Python its performance does not suffer the degradation associated with the language due to advanced batching techniques used for drawing sprites and animations.

Kivy Engine

Kivy (https://kivy.org/#home) is an open source Python library that expedites the development of applications that make use of user interfaces that are novel, such as multi-touch apps. Kivy is cross platform and runs on Linux, Windows, OS X, Android, iOS, and Raspberry Pi without any modifications to the source code.

Kivy is free to use, under an MIT license and is a community project, led by professional software developers. The graphics engine is built on OpenGL and GPU acceleration is

supported to boost performance. For performance reasons, many parts are written in C. The framework is stable and well documented in order to support less experienced developers.

Delta 3D Engine

Delta3D is an open-source gaming and simulation engine which is primarily used for *training game applications*. It is licensed under the standard Lesser GNU Public License (LGPL). Delta 3D is actually a unifying layer that sits atop of several open source projects (Darken et al., 2005). Delta 3D can also coexist with commercial software.

Delta 3D has a high-level, cross-platform C++ API that runs on Windows and Linux. This API is designed to accommodate both experienced developers who can use lower levels of abstraction and novices who can develop content through the level editor. Developers can write Python script either to the Delta3D API or to the underlying tools. The engine is completely modular and allows swapping of modules when better options become available.

Delta3D handles networking and includes record and playback capabilities for recording and replaying gameplay, an object viewer, a particle editor, a binary space partition compiler, and a runtime debug GUI. It also uses advanced algorithms to render continuous levels of detail for terrain so that it effectively renders.

Delta3D is used for several training game applications, often in military contexts. For example, Delta3D was used in an application that trains Marine Corps forward observers and teaches them how to call for and adjust artillery fire (McDowell et al., 2006). Another example is a prototype training application where sailors learn how to fight on board fires (McDowell et al., 2006). Delta 3D inherited the developer communities of many of the open source projects within it.

Cocos 2D Engine

Cocos2d (http://python.cocos2d.org/) is an open source framework used for building games and other (potentially interactive) applications with a graphical user interface. Cocos2d is distributed under a BSD license and relies on Pyglet besides Python.

Cocos2d simplifies the management of flow control, facilitates the switch between different scenes, supports fast and easy loading of sprites (e.g., images) and linking sprites to actions or effects. It also includes a particle system, tiled maps and collision detection. Additionally, it has built in classes for menus and text rendering. Cocos2d hardware acceleration is based on OpenGL. The framework is well documented, with programming guides and video tutorials.

Panda 3D Engine

Panda3D (https://www.panda3d.org/) was developed by Disney's VR Studio for their Toontown Online game, a 3D massively multiplayer online game for children (Mine et al., 2003). The engine was released as free software in 2002 and has since been developed as an open source project. Carnegie Mellon University's Entertainment Technology Center is also involved in the engine's development as are contributors from around the world. As its name suggests, Panda3D is a 3D engine, and more specifically a library of subroutines for 3D rendering and game development. The library is written in C++ while the scripting language is Python. This means that game development consists of writing a set of Python commands that invoke library functions. Panda3D comes with installer packages for Windows, macOS

and Linux. All Python versions starting with 2.6 up to the most recent (3.7) are supported and the only external dependency is a working graphics driver.

Panda3D is released under the Modified (Revised) BSD license, a permissive free software license with very few restrictions on usage. Although the engine core is completely free some of the included third party libraries are not free software and have restrictions in use, especially for commercial games. Panda3D was created for commercial game development and as such, it is a powerful engine with many tools: scene graph browsing, performance monitoring, animation optimizers, built-in physics engine and particle system and a simple artificial intelligence library. Panda3D also facilitates the debugging of game code and is tolerant of faults that may cause crashes in other engines. Panda3D provides out of the box support for advanced rendering techniques (normal mapping, gloss mapping, cartoon shading and inking) and allows developers to write their own shaders. These features facilitate rapid game development but require advanced programming knowledge. Hence, Panda3D is not a tool for beginners or amateurs.

3.3 Comparative analysis of Python game engines and libraries

In this section we attempt to compare the game engines that provide scripting support for Python. The factors selected for comparing the game engines were based on the existing literature reviewed in section 3.1. Certain factors were left out because they were a common feature of all engines, namely the scripting language (Python) and the license (open source). The list of comparison factors emerged as a result of the available and usable information for Python game engines combined with the most popular features we encounter in game engine comparison papers. We considered a feature supported if open source extensions, modules or plugins for the engine are available to support it. This is indicated in Table 1 by an "Extension" label.

Table 1. Comparison of Python game engines

	pygame	**Pyglet**	**Kivy**	**Delta 3D**	**Cocos 2D**	**Panda 3D**
Graphics	2D	3D	2D	3D	2D	3D
Level/ World editor	Extension	X	Extension	✓	✓	✓
Texture editor	Extension	✓	✓	✓	✓	✓
Physics	✓	✓	✓	✓	✓	✓
Networking	✓	✓	✓	✓	✓	✓
Mobile devices	✓	No	✓	✓	✓	✓
Written in	Python, C, Cython, and Assembly	Python	Python/ Cython	C++	Python	C++/ Python
Programming experience	Low	Low	Medium	High	Medium	High
Cross-platform developer environment	✓	✓	✓	✓	✓	✓
License	GPL	BSD	MIT	LGPL	MIT	Revised BSD

As the authors in the comparison studies presented in section 3.1 all note, ranking game engines is not a trivial task or even a good idea at all, as there is no single best game engine but rather an engine most suitable for the task at hand. In this sense, Table 1 is not meant to provide a score for each engine. If any conclusions are to be drawn, the first one is that the two most determining factors when selecting a game engine are the type of graphics needed for an application and the developer experience or else coding ability. Most other features have significant overlaps between engines, and this is generally reasonable, given that we limited the selection of engines to those using Python as a scripting language and having an open source license. Of course, if the target deployment platform is not supported by the engine, we have another eliminating factor in the selection process.

3.4 Selection of a Python game engine for novices

Given the fact that the goal was to develop a simple 2D board game in Python, suitable both for educational applications in mathematics but also suitable as a prototype for teaching programming to students with limited or no experience in Python and games programming, pygame was a rational choice. It provides the basic tools without requiring previous knowledge of game building and has an excellent set of tutorials and demos that cover all needs of a complete novice. Furthermore, one can build a game on pygame in stages, by first completing its core logic, then designing the game board, then user interaction etc.

The main features of pygame are:

- Building games with pygame is easy and fun. Even with limited programming experience, one can jump right into building simple games.
- pygame does not require OpenGL for rendering graphics. This lack of dependency is important because of several compatibility/installation issues that plague OpenGL on many platforms.
- Easy and efficient use of multi-core CPUs.
- Core functions implemented in C (10-20 times faster than Python) or assembly (100 times faster than Python).
- Compatible with virtually all operating systems and fully portable. Pygame can also be used on hand held devices and game consoles.
- pygame's core is small and deliberately simple for easy maintenance. Additional functions are developed separately as external libraries.

Pygame provides functions for creating programs with a graphical user interface instead of a command line interface so game windows contain images and colors. Every game in pygame contains a main game loop that runs until exited and has the following three functions (Sweigart, 2012):

1. Event handing
2. Game state update
3. Game screen update

The game state is essentially the set of game variable values. Indicative variables are the players' health and positions, marks made on the game board, player scores, current player, etc. Any change in a variable results in a change in the game state. For example, if a player sustains damage his health value is lowered. Game state can be "frozen" when the game is paused and/or saved when the game is stopped.

Game state is typically updated by the game main loop in response to events (for example a mouse click) or the passage of time, which is checked many times per second. Changes in the

game state may correspond to changes in the game screen. For instance, if a player uses the arrow keys to change his/her location, the position of the avatar on the screen will also change. The game loop in pygame is depicted in Figure 1.

Figure 1. Game loop in pygame (Sweigart, 2012)

4. The PyDiophantus maze game

4.1 Gameplay and educational goals

PyDiophantus is an educational game prototype that was named after the programming language Python and the ancient Greek mathematician Diophantus that was prominent for his work on algebraic equations. PyDiophantus aims to serve as a teaching aid in mathematics and particularly in the teaching of arithmetic expressions, their evaluation and the priority of operators. For instance, it can be used by students at home for practicing or in classroom to check student progress and understanding of operators and operations at any level, since it is configurable.

The game is single player and takes place on a board that works like a maze. The player starts from the top left block and has to navigate to the bottom right block in order to win the game. The player initially moves one block at a time, like a rook would in chess (only horizontal and vertical movement). Diagonal movement is optionally enabled following a number of consecutive right answers. Every destination block contains a mathematical expression whose value the player is required to enter (Figure 2). If the answer is correct, the player can proceed to his/her next move unobstructed (Figure 3). If a wrong answer is provided, a wall is erected in one of the four sides of the current block (on the side corresponding to the selected direction of movement) that prohibits movement to the particular direction (Figure 4). Every time the player answers correctly, the amount of blocks s/he can traverse in a single move (offset) is increased by one. Conversely, a wrong answer decreases the offset by one until it reaches its minimum value which is also one.

When there is one block for winning the game the FINISH block is highlighted in green (Figure 5). If the player manages to finish the game (Figure 6) the FINISH block is highlighted in red (Figure 6). If the player gives multiple wrong answers, s/he can be boxed in a certain board tile with no more possible moves (Figure 7). In this case, the player is informed that s/he has lost and all player actions cease to be processed. After the end of the game, regardless of whether the player has won or lost, a “Play again” button appears on the board and the player can start a new game with the current game configuration.

Figure 2. The player is required to compute the value of a mathematic expression.

Figure 3. The player gave a correct answer and the current offset is 2.

Figure 4. The player gave a wrong answer.

Figure 5. The player can win at the next move.

Figure 6. The player has won the game.

Figure 7. The player has been trapped and lost the game.

4.2 Game configuration

Several parameters related to the game are configurable via a plain text configuration file (config.py) included in the main code directory. These parameters are related both to the size of the board but also to the number of operands included in expressions the players are

required to calculate, the range of numbers used (in digits) and the operators that will be randomly selected in the expression. The game configuration file contains the parameters presented in Table 2.

Table 2. Game configuration parameters

Parameter	Role
ROWS	Number of board rows.
COLUMNS	Number of board columns. Because of the way the walls (horizontal and vertical) are implemented in the current version of the game, the number of columns must be equal to ROWS+1. Other than that, there are no restrictions on their values.
NUM_OPERANDS	Number of operands that will be included in the expression.
USE_SUBSTRACTION	Indicates whether subtraction will be included in the expression. The only operation that is obligatory is addition and all others are optionally switched on or off.
USE_MULTIPLICATION	Indicates whether multiplication will be included in the expression. For instance, for elementary school children who haven't been taught how to multiply numbers, it can be turned off.
NUM_DIGITS	The number of digits of operands included in the expression (determines the range of numbers that will be used).
ALLOW_NEGATIVES	Indicates whether negative numbers are allowed.

Moreover, the file includes some parameters that could be used in a future extension of the game for indicating whether division (USE_DIVISION), powers (USE_POWER) and parentheses (USE_PARENTHESIS) will be included in the expression.

In Figure 8 the configuration file and the corresponding board are presented for a 5 × 6 board and mathematical expressions including 3 operands, using subtraction, multiplication (and addition), 3-digit numbers, with negatives allowed.

```
ROWS = 5
COLUMNS = 6
NUM_OPERANDS = 3
USE_SUBSTRACTION = True
USE_MULTIPLICATION = True
USE_DIVISION = False
USE_POWER = False
NUM_DIGITS = 3
ALLOW_NEGATIVES = True
USE_PARENTHESIS = False
```

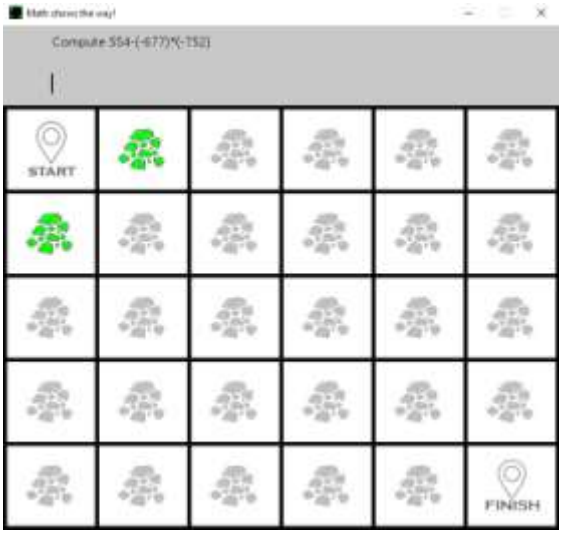

Figure 8. A sample configuration for a 5 × 6 board.

5. Utilizing PyDiophantus for learning game programming in Python

The *PyDiophantus maze* game consists of approximately 500 lines of code and implements several programming concepts/constructs and their implementation in Python, as well as game programming concepts/constructs using the free and open source library pygame.

In Table 3 we present the programming concepts and constructs that are used in the game. From Table 3 it is clear that the game implements most of the programming constructs presented in an introductory programming course.

Table 3. Programming concepts and constructs used in the game.

Programming Concept/construct	**Example**
Variables	Several variables are needed for keeping updated the state of the game. For example, variables are needed for keeping the values from the configuration file, the current position of the player, the potential next move/tile etc.
Expressions & assignments	Several expressions and assignment statements are needed throughout the game for updating the variables keeping the state of the game and the player. For example, assignment statements are used for assigning to variables the results of the calculations (expressions), the screen width and height based on the board dimensions (read from the configuration file) etc.
Conditional statements (if, if/else)	Checking the values read from the configuration file and updating appropriately the corresponding variables with the allowed operations, number of digits for the numbers used and so on. Checking for potential future moves/tiles based on current location of the player, wall obstacles, offset etc.
Loop statements (for, while)	A for loop is used for drawing the maze on the board. A while loop is used for implementing the main game loop.
Random numbers	Generating arithmetic expressions by randomly concatenating numbers of given digits and operands.
Data structures (array)	Arrays are used for keeping the potential future tiles as physical coordinates on the board and as matrix coordinates, erected vertical and horizontal walls, allowed operators that are used on the expressions that the player is asked to evaluate etc.
Functions	Several functions are used, such as: a function that returns the arithmetic expression and the correct result; function that displays a message on a given position; function that places the tiles on the board; function that places objects on the initial game board etc.
Event handling	The player clicks on the next tile, the play again button etc.
Classes	A class for adding buttons.
Importing libraries and classes	The pygame library. The TextInput class by Silas Gyger, downloaded from https://github.com/Nearoo/pygame-text-input under the MIT license. This class lets the user input a short, one-line piece of text at a blinking cursor that can be moved using the arrow-keys.

Besides the aforementioned programming concepts, several predefined pygame objects along with their properties and methods were used in the game in order either to modify important game parameters, show/edit the game window and event handling. Table 4 summarizes the pygame objects and their methods used in the game as well as their role/function.

Table 4. Pygame objects and methods.

Object / role	Methods / function
Pygame main pygame object	**init** Initialize all imported pygame modules **quit** Uninitialize all pygame modules
pygame.time pygame module for monitoring time	**Clock** Create an object to help track time
pygame.display pygame module to control the display window and screen	**set_mode** Initialize a window or screen for display **set_caption** Set the current window caption **set_icon** Change the system image for the display window **update** Update portions of the screen for software displays **quit** Uninitialize the display module
pygame.image pygame module for image transfer	**load** Load new image from a file
pygame.draw pygame module for drawing shapes	**rect** Draw a rectangle
pygame.font pygame module for loading and rendering fonts	**Font** Create a new Font object from a file
pygame.event pygame module for interacting with events and queues	**get** Get events from the queue
pygame.mouse pygame module to work with the mouse	**get_pos** Get the mouse cursor position

The possibilities of utilizing the game for teaching programming and/or game programming are several. Specifically, the game could be used for an:

- *Introduction to programming with Python*. As already mentioned, the game implements most of the programming concepts and constructs taught at an introductory programming course irrespectively of the programming language used. Consequently, the game could be utilized in such a course in various ways. For example, the students could play the executable version of the game and get familiar with it, and then the game could be gradually implemented as the various programming concepts/constructs are presented. The game could be implemented either in the context of the labs offered in the context of the course or as a sequence of assignments. The game could be implemented from scratch or a skeleton of the game could be provided in order for the students to focus on specific aspects of the game that are in accordance with the programming concepts/constructs under investigation. Alternatively, the source code of the game could be presented to students and then they could be asked to extend it with new functionality as proposed by Theodoraki & Xinogalos (2014).

- *Introduction to the programming language of Python*. The game could be utilized by students that have already been introduced to programming with another

programming language, as a quick introduction to the programming language of Python.

- *Introduction to game programming*. The game could be utilized in the context of a course on game programming or serious games programming (Xinogalos, 2018) as a case study of a fully functional serious game prototype with many possibilities of extensions. Several game programming issues and concepts could be presented in the context of the game, such as: the game loop; event handling; game state update; game draw – rendering text and graphics; utilization of game libraries (Pygame); utilization of game libraries extensions. Once again the game could be implemented from scratch, a skeleton could be provided to students, or finally the source code of the game could be used for presenting to students the process of programming a (serious or not) game and fundamental game programming concepts/constructs and afterwards the students could be asked to implement specific extensions.

A potential extension to the particular game could be related to the degree of complexity of the arithmetic operations. For example, more calculations could be presented, such as powers and parenthesis. Another extension could focus on the user interface. An example of such an extension could be a menu which gives the players the option to configure the game parameters on the fly and start a new game without relaunching the application. For bigger boards, a save function could be useful and a player could be allowed to resume a game s/he has previously progressed up to a point. In case of usage in classrooms, a scoreboard function may also be included where children are rewarded for attempts and correct answers or increased complexity of games played.

Finally, it should be noted that appropriate material, such as an interactive tutorial, could be prepared in order for the game to be used as a self learning tool for an introduction to programming concepts and constructs, or an introduction to Python in the case of a person having prior programming experience, or an introduction to (serious) games programming.

6. Conclusions

Serious games are considered to be effective tools for teaching and learning various subjects to learners of all ages. Moreover, game programming has attracted the interest of students from the Computer Science and Information Technology fields of study, but also from other fields such as Education. Programming games has also been proposed as a more effective approach to introducing novices to programming and well known educational programming environments, such as Scratch and Greenfoot, have been developed for supporting this approach.

This article focuses on investigating the possibilities of teaching and learning (game) programming using Python, a programming language that has attracted the interest both of the educational community and the software market. The main goals of the study presented were to investigate if there is a Python game engine or library that can be used by novices for implementing simple 2D games in Python (RQ1), as well as if it is possible to implement a simple but yet meaningful game that can be used as a prototype by novices for learning programming concepts/constructs and game programming concepts in Python (RQ2).

Regarding RQ1 several studies have reviewed and/or comparatively analyzed game engines and libraries. However, little research has been carried out specifically for Python game engines and libraries. Six free Python game engines and libraries were comparatively analyzed using common metrics utilized in relevant work. Specifically, pygame, Pyglet, Kivy Delta 3D, Cocos 2D and Panda 3D were analyzed using the following metrics: graphics (2D/3D); level/world editor; texture editor; physics; networking; support for mobile devices; language(s) used for implementing the game engine/library; the programming experience required for using the engine; support for cross-platform development and the license used. As previous works have concluded for game engines based on various technologies, there is also no such thing as the best Python game engine or library for all purposes. However, when the target group is novice programmers pygame is a good choice in terms of the prior programming experience required and the support provided through tutorials and demos. This was verified also by using pygame for implementing the game presented in this article.

In order to investigate RQ2, a game prototype called PyDiophantus was implemented using Python and pygame. PyDiophantus is an educational maze game prototype that can be used as an additional tool for familiarizing students with arithmetic expressions and priorities of operators. The game is fully functional and consists of approximately 500 lines of code. The game utilizes most of the programming concepts/constructs taught in an introductory programming course, as well as several game programming concepts. Moreover, it can be easily extended for showcasing many more (game) programming concepts. For example, an animated sprite could be added for moving around the maze as the player progresses in the game. The possibilities of using PyDiophantus as a case study or as a project implemented by students for teaching and learning programming concepts respectively, Python or even fundamental game programming concepts and processes are several.

It is clear that it would be interesting to study what the educational and entertainment impact of the game is both on learning mathematics and game programming. This will give us the chance to draw conclusions on the effectiveness of using PyDiophantus for practicing on arithmetic expressions or learning (game) programming in Python and pygame respectively.

References

Andrade, A. (2015). Game engines: a survey. *EAI Endorsed Trans. Serious Games*, 2(6), e8.

Christopoulou, E., & Xinogalos, S. (2017). Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. *International Journal of Serious Games*, Vol. 4, Nr. 4, 21-36.

Cowan, B., & Kapralos, B. (2017). An overview of serious game engines and frameworks. *In Recent Advances in Technologies for Inclusive Well-Being* (pp. 15-38). Springer, Cham.

Darken, R., McDowell, P., & Johnson, E. (2005). Projects in VR: The Delta3D open source game engine. *IEEE computer graphics and applications*, 25(3), 10-12.

Guana, V., Stroulia, E., & Nguyen, V. (2015, May). Building a game engine: A tale of modern model-driven engineering. *In 2015 IEEE/ACM 4th International Workshop on Games and Software Engineering* (pp. 15-21). IEEE.

McDowell, P., Darken, R., Sullivan, J., & Johnson, E. (2006). Delta3D: a complete open source game and simulation engine for building military training systems. *The Journal of Defense Modeling and Simulation*, 3(3), 143-154.

Mine, M. R., Shochet, J., & Hughston, R. (2003). Building a massively multiplayer game for the million: Disney's Toontown Online. *Computers in Entertainment* (CIE), 1(1), 6.

Moreno-Ger, P., Burgos, D., Martinez-Ortiz, I., Sierra, J. L., & Fernandez-Manjon, B. (2008). Educational game design for online education. *Computers in Human Behavior*, 24(6), 2530-2540.

Paul, P. S., Goon, S., & Bhattacharya, A. (2012). History and comparative study of modern game engines. *International Journal of Advanced Computed and Mathematical Sciences, 3*(2), 245-249.

Sweigart, A. (2012). Making Games with Python & Pygame. North Charleston: CreateSpace.

Theodoraki, A. and Xinogalos, S. (2014). Studying Students' Attitudes on Using Examples of Game Source Code for Learning Programming. *Informatics in Education*, Vol. 13, No 2, 265-277.

Trenholme, D., & Smith, S. P. (2008). Computer game engines for developing first-person virtual environments. *Virtual reality*, 12(3), 181-187.

Vasudevamurt, V. B., & Uskov, A. (2015). Serious game engines: Analysis and applications. In *2015 IEEE International Conference on Electro/Information Technology (EIT)* (pp. 440-445). IEEE.

Xinogalos, S. (2018). Programming Serious Games as a Master Cource: Feasible or not? *Simulation & Gaming*, Vol. 49, Issue 1, 8-26.

Yampolsky, M., & Scacchi, W. (2016). Learning game design and software engineering through a game prototyping experience: a case study. *In Proceedings of the 5th International Workshop on Games and Software Engineering* (pp. 15-21). ACM.