Bulletin of the *Transilvania* University of Braşov
Series III: Mathematics and Computer Science, Vol. 2(64), No. 2 - 2022, 203-214
https://doi.org/10.31926/but.mif.2022.2.64.2.16

USING PYTHON IN DEVELOPING VIDEO GAMES

Dana SIMIAN [*,1] **and Adrian VULPEANU** [2]

Abstract

The aim of this paper is to investigate the capabilities of Python language to be used for game developing. On the other hand we are interested in analyzing the advantages offered by Python in introducing students and non-professionals in the game design world. As a prove of concept we implemented a slider game in Python using Ursina engine and highlighted how easy different concepts from game design can be implemented using Python.The conclusion is that the main disadvantage of Python (time execution) can be overcome exploiting the multiple data structures provided by the language, the use of OOP (Object Oriented Programming) and of multitude of available frameworks. The ease of learning and understanding the language is a major advantage. Our code can be accessed on Github. Our study and project can also be used as a starting point for developing Python-based projects for introductory game design courses.

2000 *Mathematics Subject Classification:* 68NXX, 68WXX.
Key words: Python, game design

1 Introduction

Games are becoming more and more popular over the years. The games industry has a huge turnover. Even if, according to an analysis made by Ampere Game, it is expected that the year 2022 there will be a slight decrease, it is expected that the year 2023 will again register an increasing trend. The lockdown during the COVID-19 pandemic has led to an increase in video game players. New games are presented to players every day by independent game companies and corporations, and news of game release dates appear on any social network in the form of advertisements. As a result, more programmers and developers are interested in games development and design, but the skills required are vast and varied: programming, animation, design, sound, environment, art, etc. Not everyone has these skills at hand, and that's why, by example, triple-A games

[1*] *Corresponding author*, Faculty of Sciences, Research Center in Informatics and Information Technology *Lucian Blaga* University of Sibiu, Romania, e-mail: dana.simian@ulbsibiu.ro
[2]NXP, e-mail: vulpeanuadrian1994@gmail.com

are developed by big teams (or game studios) like: Ubisoft, Naughty Dog, Square Enix, CD Projekt Red, etc. That doesn't mean that an individual can't make good games by itself. There are many examples of developers who created amazing games like Minecraft by Markus "Notch" Persson, Stardew Valley by Eric Barone, Undertale by Toby Fox etc.

Many and varied game courses are available, offering the individual possibility of those attracted to the field of game design and development to train themselves. Complex learning paths within some universities also offer courses in this field. A multitude of books try to cover certain parts of game design [1], [3].

The goal of this article is to offer an image on Python capabilities in game design and development. We want to prove that using Python can be a good start in learning the basic elements for developing a game, offering a simple and attractive way of introduction in this field. The method chosen for the demonstration is a practical one. We created a 2D game with the basic functionalities specific to slider games, as well as some additional ones using exclusively Python. Developing a 2D and not 3D game is not an impediment to our goal, because the experiment we did refers to the use of the Python language to create the bases necessary for promotion in the gaming industry.

On the other hand, creating your own computer game in any programming language is a great way to learn or improve your programming skills. To use a lot of basic programming skills, you can build more complex games like 2D games (Mario/Zelda) or even a 3D genre like low-level shooters where you can try to replicate the mechanics from Call of Duty 2. In this way you can build up your skills to use loops, conditional statements, functions, object-oriented programming, to understand Player/Npc movement on a 2D or even a 3D X/Y/Z axis, to learn the game logic features that we all have seen while playing games, like checkpoints/spawn/player life/quest etc. [4].

The rest of the article is organized as follows. In Section 2 we analyze the current state regarding the use of Python in game development, highlighting its advantages and disadvantages. Section 3 describes the main results and the design and implementation of our the game. Conclusions are presented in Section 4.

2 Python in gaming development

When we are faced with the choice of a programming language to choose to get started in game design and develop a game, the most common choices are C++ and Java. Most of the successful games in the triple A game industry are usually created with them.

C++ is a rather difficult language to learn, but it allows more direct control over hardware and graphics processes, which is very important in video game design. Java, runs on everything from printers and microwave ovens to complex video game systems. It is a very dynamic language with lots of applications, which makes it seem like a good choice to learn. Checking the most used/best programming languages for game development on Google, we can clearly see that

Python is not even in the top 5 in most sources. Over time we can noticed a lack of games created in this language (it is used, in most cases, in the scripting/data processing part [2]). If we consult the wiki page of the Python programming language, we can see, in the Video Games section, that we have a list of only 14 games.

There are a few big reasons why Python can be preferred for an introduction to game development:

- The clear and clean syntactic structure of the Python language makes it stand out from other programming languages for game design.
- Python code is easier to understand than Java or C code. Its low learning curve makes it a popular choice among game developers.
- It incorporates a variety of other languages, such as Java and C -Cpython and Jpython. Therefore, the transition from developing games in Python to developing games in other languages is easy
- Python is object-oriented, has built-in high-level data structures, and supports dynamic typing along with dynamic binding.
- There are a multitude of frameworks and libraries for Python (such as Pandas, Numpy, Beautiful, etc.).
- Supports GUI

In general, Python is used in game implementation (especially in AAA games) for "side jobs", such as addons / quests / internal scripting, while the main part of the game (main code) is done in another language (C# or Java). One of the main reasons for the low use of Python in game development is that Python was not designed to be very fast. The most popular games use compiled language such as C++ or C#, where the written code is converted into raw machine language, which is much faster. Major differences in execution times can be observed, in some cases, C# ends up being 44 times faster than Python.

We can summarize the advantages and disadvantaged of using Python in gaming development as follows.

Advantages:

1. Flexibility – a large number of application types can be created because it has a major set of support packages.
2. Portability – supports several operating systems; one can run the same code on different platforms without needing to compile.
3. The ease and simplicity of writing the code which makes the extension and maintenance of the code very easy and fast.
4. The support offered by this language from the Python community (due to large number of developers using Python).

Disadvantages:

1. There are no specific game engines, popular in the game industry, that use Python as a programming language, such as unreal engine.

2. High execution time (compared to other specific languages for creating video games such as C++ or C#)

3. Lack of protected/private classes. Although we can fight using specific conventions in the Python language the objects created by us will not be encapsulated as in other languages.

Several frameworks and/or libraries can be used to develop games in Python.The most popular framework is Pygame; it is not included in the base package, but it is free, open source and can be easily installed using the generic command

```
pip install *packet name
```

A relatively new framework is Ursina. Pygame had its first release in 2000 and since 2016 the development is no longer continued. Ursina had its first release in 2020 and is still regularly updated.

In case we want to create a game from scratch in the Python, we have other options such as Pyglet, Cocos2d, Pandas3D (focused mainly on creating 3D games).

3 Main results

We designed and implemented a 2D game, as a desktop application, to prove Python's capabilities in simple implementation of the main features required of a game, such as objects, enemies, environments, and checkpoints.

The game, called Lord Muffin, belongs to the category of slider games. It was designed to contain all the features of the specified games genre.

- Different levels.
- Multitude of enemies with different attacks.
- The character will have specific powers to defeat opponents.
- Score board.
- Coins counter.

At the same time, we also created elements that are not specific to this type of games (Mario, Zelda):

- Checkpoints
- Inventory

3.1 Chosen technologies

In developing the application and writing the code, we used PyCharm, a specific IDE for Python. PyCharm is considered a smart code editor, having fast refactoring and enabling intelisence. It has features for debugging, profiling, remote development, and code testing. It has support for popular web technologies, web frameworks and scientific libraries. Allows version control; allows automatic control and synchronization with automatic github/gitlab, it is no longer necessary to use the terminal/gitbash. We used the community version, which is free.

Ursina Engine was mainly used for the actual implementation of the game (https://www.ursinaengine.org/documentation.html). Using Ursina models, textures and code can be reloaded during the game. To create game models, .psd or .blend files are automatically imported. Easy-to-use generic classes are used for object geometry (Entity class). It offers boilerplate code and build-in animations, pre-made player models like FirstPersonControler (mainly used in 3D games) or, in the case of the current project, 2D platformer controller. Ursina is compatible with both Windows and Linux platforms. Unfortunately, it is not compatible with IOS and Android.

For designing and editing of objects (ground/attack towers/enemies/player) mainly own images or open source vector images were used. They were edited using Paint 3D from the standard package of Windows 10. We chose git as the versioning tool. All the code was uploaded using the VCS terminal from Git. At the same time we used the VCS built into the Pycharm IDE. Github was chosen as the storage medium. The game can be found at
https://github.com/VulpeanuAdrian/LordMuffinGame

To get the executable version of our game and upload it to a download environment, to become easy to download and play for any Windows/Linux user who does not have the Python package installed, we used Pyinstaller. Pyinstaller can be used on any operating system, Windows, Mac OS X, GNU/Linux, but it is not a cross-compiler.

3.2 GUI

The GUI is automatically generated using the framework elements of the Ursina library. It consists mainly of 2 different elements respectively: the Menu and the Levels. The Menu consists of generic elements found in most games such as the Play, Exit and Help buttons The second type of graphical interface found in the game is the level interface (it appears immediately after pressing the Play button). In Figure 1 we observe different elements of the level GUI, such as text label (Score), graphic labels (Health bar), as well as specific elements found in the Ursina engine. To better illustration we encapsulated all elements in coloured boxes.

The specific entities within the framework encountered in the present capture are

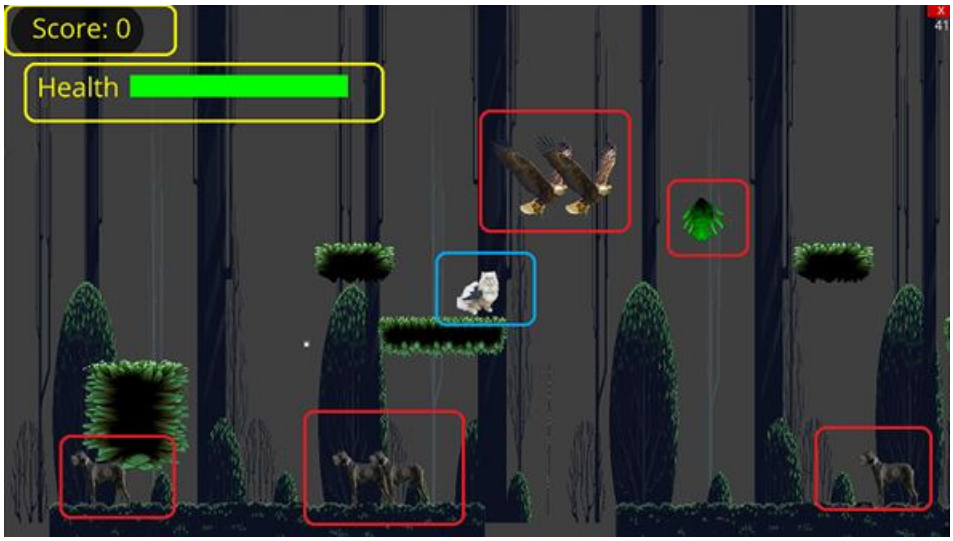

Figure 1: Level interface

- PlayerControl (the actual player in this case: the cat).
- The enemies (in the present case, the eagles, the dogs and the green plant). Within the framework these are Entity type objects. As a graphic interface level they can be seen as graphic labels with additional properties.
- The sliders, respectively the elements on which the player (the cat) is located.
- The background.
- Number label (score counter, health bar).

3.3 Game logic and objects

The logic of the game is simple, corresponding to a slider game. The central character, in our case the cat Lord Muffin, moves on the existing sliders, through different types of environment, meeting different types of enemies, the goal being not to lose his life and to achieve the highest possible score, by eliminating the enemies and by accumulating coins (points). The movement of the characters (player, enemies) is managed by specific attributes such as speed or power.

Next, we will highlight the main objects we built in the game and how Python facilitated the implementation of the desired behaviors of these elements.

The implementation of the main character (player) is done with just one line of code, using PlarformerController2d:

```
player = PlatformerController2d(y=22,x=3, scale=(1, 1,0.01/2), color=color.white,
texture=f'images/muffin_02.png',)
```

We can very easily modify the already defined class attributes of the player just by changing the default attributes of the player object. For example, in a

scenario where the player gets a bonus/extra power, we can change his speed using the speed attribute.

```
player.speed=20 # desire speed
```

There is a built-in Entity class that almost all objects inherit from. It is very useful to create some generic elements that we might need in a game such as: the background (i.e. the main theme that we can see behind the game); enemies; power ups; motion elements such as sliders, cubes, etc.

We created .py files at class level for each type of objects, such as for example enemies. This help us to reuse objects and to refactor without changes in many parts of the code. For example: the player's enemies are in the enemies.py directory. The multitude of attributes that we have at our disposal for an object makes implementation easier. For example, we have attributes for adding a graphic image associated with the object (self.texture), for coordinates (self.x, self.y), color attributes (self.color).

We have created several types of enemies:

- Terrestrial enemies (DogEnemy) and aerial ones (BirdEnemy). Upon touching these enemies, the central character will lose the game.

- Complex enemies (RaccoonEnemy). These enemies have in addition to the classic functionality of an enemy, an additional one. They shoot a cloud of smell that will remove the player from the game if he touches it. The implementation is similar to that of a classic enemy, such as DogEnemy except for the attack functionality feature.

- Static enemies of type PlantEnemy, which do not move, but the central character loses life points if he touches them.

- Static enemies of Type AttackTowerEnemy. They can be touched by the player to overcome them, but they attack with laser-type objects, similar to the raccoon's attack. If the main character is hit by such an object, he will lose the game.

- Boss type enemies (MouseEnemy). These are special enemies that require multiple attack hits to be destroyed; they are situated at the end of each level, are harder to defeat and have their own attacks.

Enemies of different types (DogEnemy, BirdEnemy, RaccoonEnemy) are added to a list immediately after each one is created.

We have also implemented a PowerUp functionality. The player can get certain powers if he eats (touches) a bowl of food. PowerUp features are essential to progress in the game, being the only possible way in which the character can defeat enemies. Should the player reach and touch such an item, he will receive unlimited power as long as he does not lose the game. In the present case, he will receive an attack in the form of a ball, and if this ball hits the enemies, they will be defeated, except for boss enemies that require several ball hits.

The implementation of the logic behind the power and attacks is done in the update method that will be overridden for each object. This is an advantage offered by OOP. More details about the update method will be presented in the Subsection 3.4.

Other objects built in the game are Coin objects and Audio objects.

One of the objectives of the game is to have a high score, which increases based on enemies defeated and coins collected. The coins created in the game are dynamic, they rotate. This effect was achieved using the rotation_y attribute from the Entity base class. If the player hits a coin, the coin disappears and the score increases. All coins are added to a list and their position is predefined according to the ground, to be possible to collect them. When the player collects a coin, a predefined sound will play using an Audio object. The audio object used is a predefined object in the library, only the path to an mp3 audio file is needed for the sound.

Implementation of audio objects is done within the constructor inside the level, so the game has different sounds when the player jumps/defeats an enemy and collects a coin. The game also has a generic song that plays continuously as long as there is no jumping/coin collecting action. This implementation was done using the audio base object from the library.

Implementation of logic, gameloop and objects management done separately to provide better code traceability and at the same time to easy modification/refactoring (need to actually modify only imported files). The game can be run from main.py. In main.py we have the first interface (Game Menu). In addition to classic elements such as Exit, Help, Play buttons we used animations. The animation in the menu is very easily done using the Ursina engine. The movement of the cat's paws on the laptop keyboard is an event created in a single line of code using the Animation class. At the same time, on the button objects, we can set actions automatically using the on_click = function_name argument, thus we can easily create a function that will start the first level, set this function on the start button, so when the button start will be pressed, the game will start. Similarly, functions are used for the information found in the Help menu and the Quit function of the application. Initially, an Ursina instance is created, after that all objects (buttons, entities, levels, etc) are created and finally the run() function is called. This syntax is also common in other similar libraries such as Pygame.

The multiple data structures that Python has are an advantage in implementation. By example, using tuple data types allows you to perform swaps in a single line of code.

3.4 Update functions

The Update function describes the logic behind each object. This function is inherited and overridden depending on the object type. The object inherits the base class Entity from the Ursina framework. By overloading the update function (within the Entity class) we can control that object.

All enemies are stored in a list, ensuring easier traceability in case the player

gets hit or touches one of the enemies. In the Update function we will go through the list and check each time if the player's position on the X/Y axis will intersect with the position of one of the enemies.

3.4.1 RaccoonEnemy Update function

Within RaccoonEnemy object we have a similar implementation to that of a classic enemy (e.g. DogEnemy) except for the attack functionality. The function Update has the role of controlling the attack of the RaccoonEnemy, having several objectives:

- In the first instance, it will be asked if the object is visible (the attack takes place only at a predetermined distance)
- If it is visible, the axis will also increase by 0.10 units/axis and decrease by 0.10 units, thus an attack will be obtained (a movement of the object diagonally). Within the frame, the sizes are standard, the way to obtain these values used here and in other cases, such as player speed, other enemies' attacks, enemies' movement were determined empirically after several trials.
- If the attack (the cloud) will move 5 units on its initial x/y axis (initially at its creation), it will disappear and start again from its initial position.

We also need to implement the logic behind the attack, i.e. what will happen if the player is touched by the enemy/his attack. In this case, in the Update function of the first level class inside a for loop in which we have all the enemies, we have the following flag:

1. The type of enemy is initially checked (because each enemy has different rules-types of attack and damage). With the help of a classic function from the Python standard library isinstance, we actually ask ourselves if the enemy is of the Raccoon instance type.
2. If this flag is True and the player's position is close (under 1 unit) to the enemy or their attack, the player will lose the game.
3. In this case, the player character will rotate 90 degrees, a game-ending trigger (self.switch) will activate and the life bar will turn all red. (This kind of loss differs from an enemy to another, in the case of other plant-type enemies, the player will gradually lose only life, not the game.itself)
4. This logic is used within the Update function inside the first level, this function being implemented within the framework, thus we can implement any type of event (action) within it.

In Figure 2 is illustrated the effect of RaccoonEnemy Update function

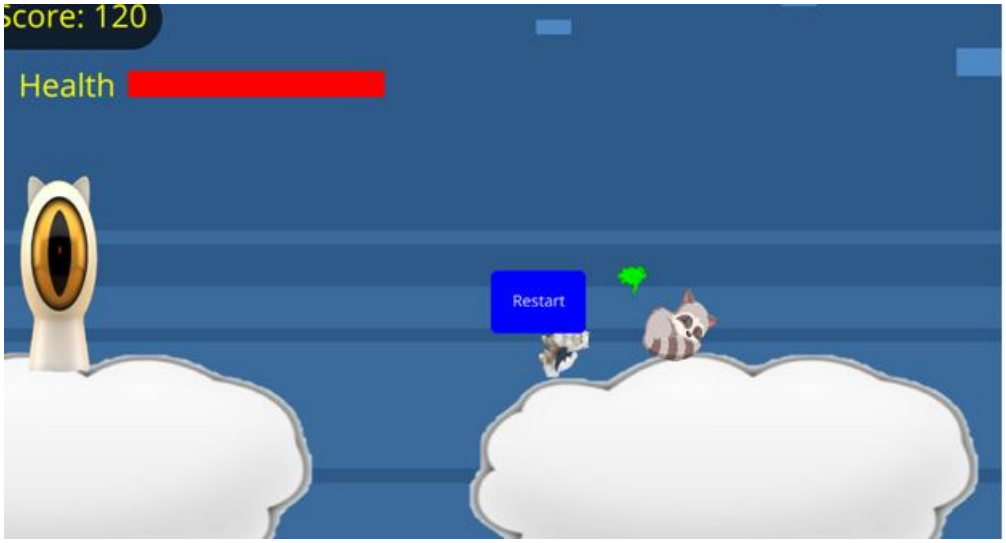

Figure 2: Player dead at raccooon attack

3.4.2 PlantEnemy Update function

In the case of a PlantEnemy if the player touches it, he will lose life points. In this case, in the Update function of the first level class inside a for loop we will perform the followings steps:

1. We will go through a list of all the traps (plant-type enemies).
2. If the player is in the attack range of the plant, he will lose life points from the green bar situated on the GUI.
3. If the life bar reaches 0, the game will be over.

3.4.3 AttackTower Update function

Due to the refactoring and using Entity as the base class we will have a similar implementation with the RaccoonEnemy. The only major difference is in the update method, where we only need to decrease the x-axis by 0.05 units to produce the attack movement towards the player. When the distance of 10 will be exceeded, the attack will disappear and start from the initial position.

In this attack, a predefined texture in the framework was used

$$\text{texture} = \text{arrow right}$$

We did not create and edit a graphic object as in other cases to show and use predefined objects in the library.

3.4.4 MouseEnemy Update function

The implementation of Update function for MouseEnemy, is similar as in other enemy type objects (RaccoonEnemy). The major difference is that, unlike the

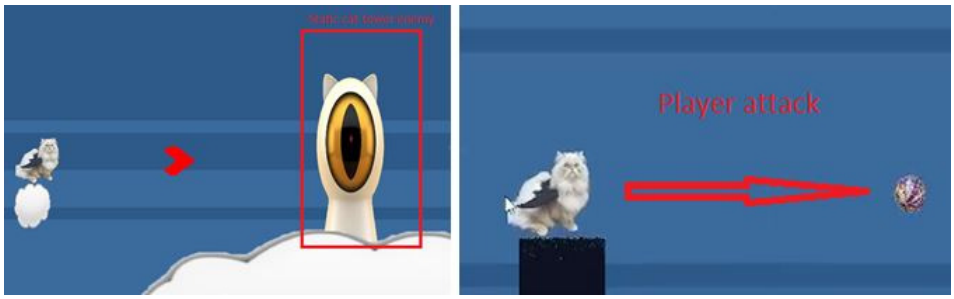

Figure 3: Example of Tower attack (left) and player attack (right)

rest of the enemies, it will not die in the case of being hit only one only time. It needs 5 hits. For better traceability within the MouseEnemy object, an attribute self.mouse_hit_points was set initially equal to 5. This attribute represents enemy's life. So the player needs 5 hits to defeat him. In the case of each hit, the attribute decreases by one unit. Also after each hit, the enemy will become invisible for 0.25 seconds, using the invoke function in which we specify which attribute change we want (visible/invisible) and delay (for how long to have that change).

3.4.5 PowerUp Update function

In case the player gets a power, a flag with that power will be set, so we know that the player has that power as long as he doesn't lose the game. In case the flag becomes true (the player gets the power), he can press the R key to launch an attack. In the implementation we have to specify from this side the attack will start (left or right), that's why we use other 2 flags. Also, the object of attack (in the present case: the puck) had to know the player's place (the puck goes from the right/left player to an area on the left/right depending on his place), that's why he needs the player position. The attack object, the ball itself, inherits from the Entity base class as well as the rest of the element, the only difference would be the attack method created for our object to receive the player's new coordinates. In Figure 3 illustrate a Tower attack (in the left) and of player attack (in the right).

4 Conclusions

In this paper we investigated the gaming development capabilities offered by Python language. We wanted to prove that Python can be a good option for beginner to familiarize with the basic elements needed for game developing. Also, we wanted to prove that you can easy develop a game (not very complex at the beginning) using exclusively Python. Even if one main disadvantage of using Python is higher execution time, we can optimize the execution time by making full use of OOP, structuring the code as efficiently as possible and at the same time using frameworks. As prove of concept we developed a slider game using

Python and Ursina framework. The game is named Lord Muffin and whole code is available on Github at the address

https://github.com/VulpeanuAdrian/LordMuffinGame

The conclusion is that Python can be used with confidence for any project due to the multitude of libraries available for any need and at the same time the ease with which this language can be learned. Additional artificial intelligence features can be easily and efficiently introduced using the multiple libraries that Python has in this area.

Acknowledgement: The first author was supported from the project financed by Lucian Blaga University of Sibiu through the research grant LBUS-IRG-2022-08.

References

[1] Nystrom, R., *Game programming patterns*, Genever Benning, 2014. A visual step-by-step guide to writing code in Python

[2] Sayeth Saabith A.L., Fareez M.M.M. and Vinothraj T., *Python current trend applications - an overview. Popular Web development in Python*, Int. J. of Advanced Engineering and Research Development **6/10** (2019), 6-12.

[3] Schell, J., *The art of game design: a book of lenses*, Rouledge, Taylor and Francis Group, Second Edition, 2022. A visual step-by-step guide to writing code in Python

[4] Sweigart, Al., *Invent your own computer games with Python*, No Starch Press, US, 2016.