MASTER OF SCIENCE IN APPLIED SCIENCES AND ENGINEERING: APPLIED COMPUTER SCIENCE

Vrije Universiteit Brussel

Advanced Programming Concepts

Prof. Bart Jansen, Maryna Kvasnytsia, Taylor Frantz
and Redona Brahimetaj

Physics Sandbox

Madhurima Khamroy - 0599148
Miriam Nohemi López Cruz - 0596754
Samarzeit Muralidharan - 0597253
Tania Fernanda Ugarte Guzmán - 0599620

TABLE OF CONTENTS

I. OBJECTIVE

The aim of the project is to present a GUI where the end-user can draw 2D shapes, and to simulate and visualize basic interactions (such as collisions and gravity) at play.

II. FUNCTIONALITIES

1. How to make it run?

- **Copy the files:** SandBoxGame.py, Bakground.jpg, Rules.jpg on your computer.
- Run the file SandBoxGame.py in your code editor of your preference or in the terminal in your preferred environment of python.
- A window as in Fig. 1 will appear.

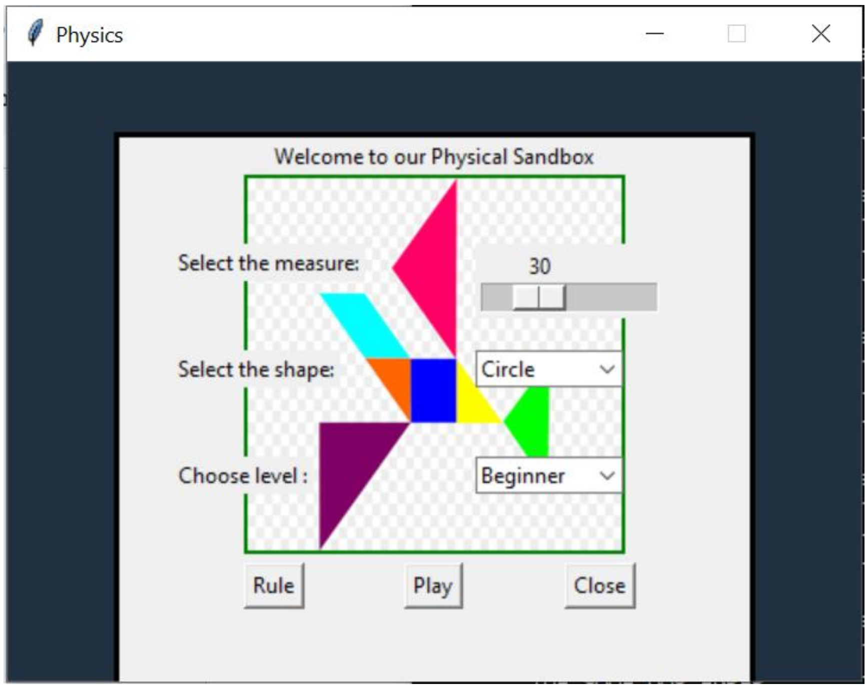


Fig. 1. Initial Window of SandBox

2. How to fill this form?

Let see in the next table:

Option	**Functionality**
Select the measure (Scale value):	The user is able to choose between 20 and 60, which will be the size value of the main shape to play the game and move with the mouse.
Select the shape (Combobox value):	The options are: 'circle', 'triangle', 'square'. The users will choose the shape that they want to use to play as a "main shape" to move with the mouse.

Choose level (Combobox value):	The options are: ‘Beginner’, ‘Intermediate’, ‘Advanced’. The user will choose the mode of the game, it means that the game will have different difficulty which is controlled inside the code changing the values defined by pymunk. Across the level is increasing the size of the shapes will be bigger.
Play Button Play	It will send the data input to run the game, so the main gaming window will be displayed as it is shown in Fig. 2.
Rules Button Rule	It will show a small window with the rules of the game.
Close Button Close	It will close the window.

Table 1. Functionalities of the initial window

3. How to play?

Pressing Play Button:

After pressing this button Play. The window game is displayed (see Fig. 2), where the user will be able to move the main shape (the one that he chose before in Fig.1), by hovering the mouse. The main goal is to avoid collisions with the small shapes that are appearing on the screen.

The small shapes coming in the screen are triangle, square and circle, they have been continuously created in a random order and they are coming from random places in the screen.
On the right top of the screen the accumulated score is shown:

- Circle scores 3 points.
- Square scores 5 points.
- Triangle scores 7 points.

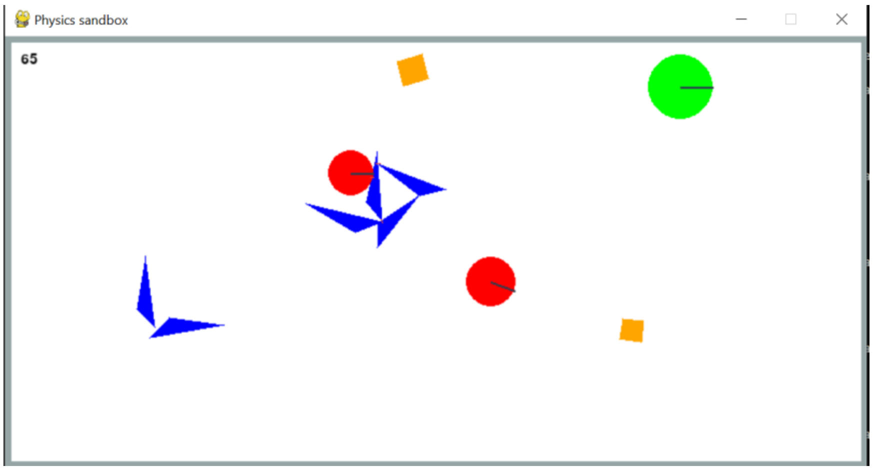
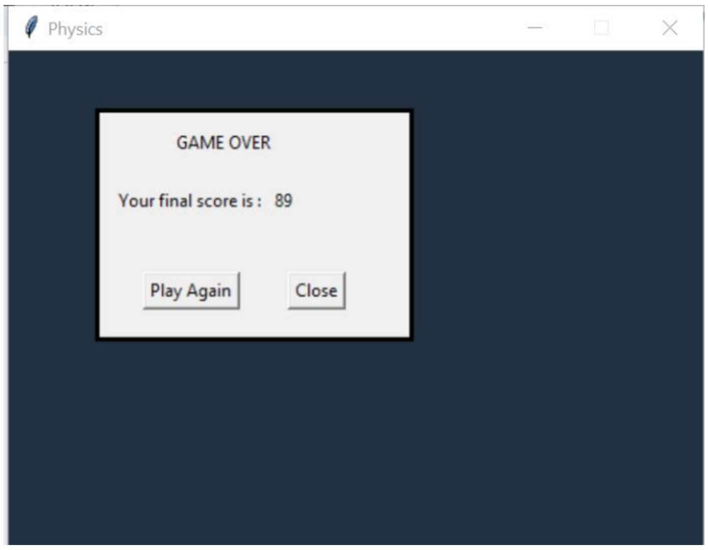


Fig. 2. Game window

Game over:

When the main shape makes a collision with any of the small shapes, the game is over, so the user will see a small window with the total Score gained and he will have the option of playing a new game or closing the screen and quitting from the game (See Fig.3).



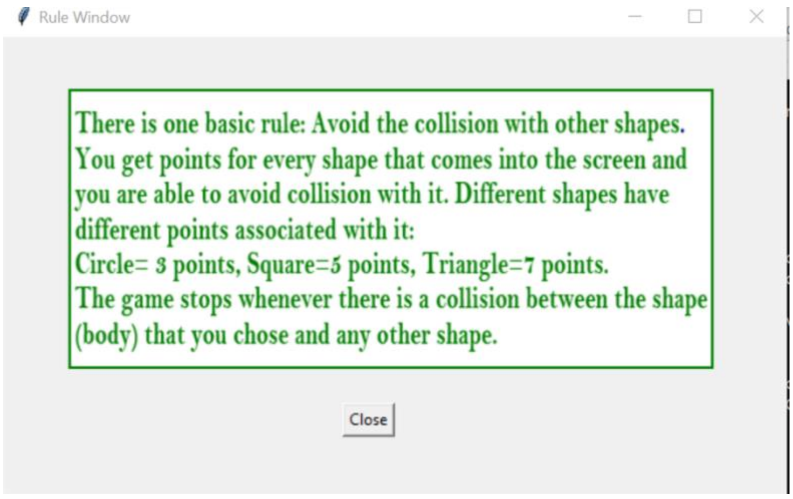
GAME OVER

Your final score is : 89

Fig. 3. Game over window

The rules:

By pressing the button "Rule" (see the option in Fig.1), the user can see what the basic rules of the game are (Window of rules in Fig.4). By pressing close the user can go back to the initial window with the initial form to start the game (Fig. 1).



There is one basic rule: Avoid the collision with other shapes. You get points for every shape that comes into the screen and you are able to avoid collision with it. Different shapes have different points associated with it:
Circle= 3 points, Square=5 points, Triangle=7 points.
The game stops whenever there is a collision between the shape (body) that you chose and any other shape.

Fig. 4. Rules

III. DESIGN CHOICES

1. Object Oriented Programming Approach

The base class of our project is the *CreateBody* class, where we are defining a Pymunk Body, which must have a mass, moment, type (if it is static or dynamic), position (based on the position of the mouse) and a velocity. This class has an update method, which is responsible for the bouncing when the object hits the wall.

The three classes namely: *CreateBox*, *CreateCircle* and *CreateTriangle,* inherit the attributes and the method from the *CreateBody* class to create the Shapes for the Body. In these three classes the respective shapes are created. These three shapes are assigned with different colors.

The Game class can be instantiated with three arguments namely: radius, type of the shape the player chooses to use and difficulty level the player chooses to play in. The *generate_random_objects()* method generates the obstacle objects on the screen. Based on the difficulty level, the size and velocity along two axes of random objects are generated, and the elasticity, friction, mass and collision type are defined for that shape. This method will return a value random which will be used later to update the score.

The run method is the main method. Here we are initializing the pygame and then creating the screen, clock and space. The *collision()* function is defined to set the class variable *is_collision_detected* to True if collision is created and the collision handlers are defined. Now the four walls are created and to make the game a bit harder in the advanced level, the elasticity of the walls are increased to make the objects bounce faster. Then comes the while loop responsible for the whole game. In this, first the random objects are created after a certain time interval. As mentioned earlier, the return value from *generate_random_objects* method is used to update the score of the player and every shape has a different score. Then, based on the type of shape chosen by the player the respective shape is created, and the color, collision type and elasticity are defined. When collision is detected, the while loop breaks. If collision is not detected the simulation is updated and the loop continues.

The final class in Window class. While this class is instantiated both window and frame are created. In the draw method the labels, text field are created to get the inputs from the player like size of the player object, difficulty level and many more like shown in Fig. 1. Then we have the rule method which will display the rules of this game in a separate window like shown in Fig. 4. Then we have the play method which calls the run method in the Game class. Finally, the *draw_window_over* method displays the score of the player and asks if the player wants to play again or quit like shown in Fig. 3.

Finally, we have a main function which is not part of any class which calls the draw method in the Window class. And then we are calling the main function.

2. Packages and libraries

The libraries we made use of for the execution of the project are detailed below:

- Pymunk:

Pymunk is an easy-to-use pythonic 2d physics library that can be used whenever you need 2d rigid body physics from Python. It is built on top of the very capable 2d physics library Chipmunk.

Three of the basic classes of Pymunk are:
Rigid body: it holds the physical properties of an object. (mass, position, rotation, velocity, etc.) It does not have a shape by itself.
Collision shape: an around the body that can collide.
Space: It is the basic simulation unit in Pymunk. It is the area where physics is being calculated. We can add bodies and shapes to it.

- Pygame:

Pygame is a set of Python modules designed for writing video games. Pygame adds functionality on top of the excellent SDL library. This allows to create fully featured games and multimedia programs in the python language.

- Tkinter:

The tkinter package ("Tk interface") is the standard Python interface to the Tcl/Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, including macOS, as well as on Windows systems. We have used Tkinter to create the interactive windows (and the elements in it) for the game.

IV. GROUP WORK

Initially, all the members of the team gathered to discuss some initial ideas that each one of us had to implement the sandbox. Some video tutorials about simulating physics with pymunk and pygame [4] were shared by Samarzeit, and that gave us a better understanding of how these libraries work together.

Tania and Miriam worked on an initial version of the game. Tania focused mostly on the game design, i.e. how could the simulation be more dynamic and interesting from an user perspective; she also included the tkinter Window Class. Miriam thought about the modular approach of the code and the pymunk implementation.

Samarzeit merged both initial versions and added extra functionalities with pygame. Madhurima kept working in the tkinter Window class. Both complemented the Game Class. At this point we submitted our intermediate report.

After the intermediate report was delivered, we agreed upon some final changes on the design of the game: we decided to remove the "Learning" and "Gravity" modes and instead we added extra functionalities to the Game mode, such as the level of difficulty.

The last part of our workflow consisted in the debugging and optimization of the code. This was treated as a cyclical process in which one of us analyzed the code trying to fix some errors and code redundancy, then she or he "passed" the code with the improvements to the next person, who did the same, and so on.

V. CONCLUSION

The libraries from python are useful to implement in games in an easy way, by calling their native functions, but it is important to create a specific structure from the beginning in order to make them work together correctly.

We tried to avoid hard coding as much as possible, but in some places like elasticity of the walls and physical properties of the random objects, we hard coded it because we faced few errors like the objects getting accelerated too much after colliding with walls or even other objects which makes them move in abnormal way in screen and hinder the flow of game. So we hard coded a few parts.

The modular design of the code allows for easy extension and customization, being this one of the main advantages of Object Oriented Programming.

We have done the testing thoroughly, where each part of the code has been tested by all team members. It is common to overlook our own errors when coding. So, to prevent this, the code written by one team member has always been tested by others as well.

VI. REFERENCES

[1]*Pymunk — Pymunk 6.4.0 documentation*. (n.d.). https://www.pymunk.org/en/latest/
[2] *GettingStarted - pygame wiki*. (n.d.). https://www.pygame.org/wiki/GettingStarted
[3] *tkinter — Python interface to Tcl/Tk*. (n.d.). Python Documentation. https://docs.python.org/3/library/tkinter.html
[4] Clear Code. (2020, June 2). *Simulating physics in Python* [Video]. YouTube. https://www.youtube.com/watch?v=YrNpkuVIFdg