Programming IV Project:

Pac-Man in Python

Juan Pablo Ospina Bustamante, Kevin Alexander Moreno

*Risaralda, Universidad Tecnológica de Pereira, Colombia*

Emails: [juanpablo.ospina@utp.edu.co](mailto:juanpablo.ospina@utp.edu.co), [kevinamh24@gmail.com](mailto:kevinamh24@gmail.com)

***Abstract:* The famous game, Pac-Man, is recreated using the programming language Python and one of its modules, Pygame. A clear notion of the objects that might be used is described, along with their functionalities and interconnections. Five design patterns are implemented: Factory Method, Iterator, Builder and Singleton.**

**Keyword: Builder, class, design Pattern, factory, ghosts, iterator, Manhattan, objects, observer, Pac-Man, Pygame, Singleton.**

**1. INTRODUTION**

In the learning process of the object-oriented paradigm, in comparison with other paradigms such as the imperative paradigm, difficulties may arise now in which projects are designed and codified. This is due to the different ways that exist to plan a project and carry it out.

It is important to be clear about the objects you want to create and the interconnections they can have with each other. Using a class diagram, it can be easily seen represented.

By using design patterns, a problem can be solved by using templates that design patterns provide.

These templates describe the solution to a recurring problem at the time of project coding. However, these patterns are not solutions that can be implemented directly in the form of code to the different programs that you want to create.

In this way, you must also have great clarity to implement them, maximizing the benefits they provide when they are used.

The formalization of these patterns may require additional efforts from the programmer; however, these efforts imply a great gain in terms of cost and benefit due to their reusability.

In this way, it is possible to think and plan the ways of developing a game, such as Pac-Man, in which design patterns are used, and which is carried out using the paradigm of object-oriented programming, where each element of the game can be represented by objects that

would perform a specific function that would make up the project that was carried out.

**2. DEVELOPMENT OF PAC-MAN USING DESIGN OBJECTS AND PATTERNS**

**2.1 Design:**

Ensuring a clean execution for the project, we proceed to design the basic structure that would make up the project. This design would cover different parts of the game, including:

● The labyrinth

● Pac-Man

● The Ghosts

● The Points (Pac-Dots and Power Pellets)

● Sounds and Score of the game.

● Player movement and ghosts.

In the process of creating the labyrinth, the first problem is encountered, regarding the easy creation and modification of the labyrinth. On the other hand, the representation of this on the screen would consist of a group of objects different from each other that would make up the types of walls necessary to recreate the labyrinth of the original game.

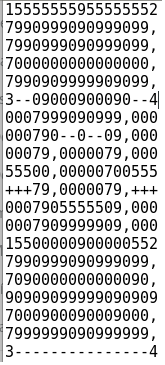
There are three types of walls that need to be rendered in Pac-Man. Horizontal walls, vertical walls and filled walls (wall blocks); Each of the walls can have a specific position where a pixel needs to be up, down, left or right. There are also the corners of the labyrinth, which are represented by two objects at the same time.

Because there are many wall types that can be created, the first design pattern is used. The Factory patterns.

Using the Factory pattern, it is possible to send a series of attributes or necessary conditions to a class, and this would return the desired object.

It is also necessary to represent the labyrinth by symbols that can be easily read and modified.

The representation and construction of the labyrinth is then used using a text file (txt), in which the walls would be represented by different symbols, as can be seen below:



Where:

0: Represents an empty space.

1,2,3,4: Represent corners.

5, +, -: Represent Horizontal Walls.

7: Represent Vertical Walls.

9: Represents Wall Blocks.

In this way, the labyrinth is created, which can be easily modified through a text file.

In order to locate the Pac-Dots points on the screen, it is necessary to know what free spaces the labyrinth has in which the points could be positioned.

This would mean generating a considerable number of points (about 240), which would be complex to generate manually, so the second design pattern is implemented. The Iterator pattern, which is used to iterate over the labyrinth, and that for each iteration delivers a point-type object, where possible, which would be grouped in a list and then interact with them.

Since the Power Pellets are in a static position in the maze (and there are only 4), they can be created manually.

Focusing on the movement of the player, a class is required that allows the handling of Pac-Man images, and that allows his movement. That is why the Player class is created. The game runs in a while () loop, in which a keyboard read must be performed at each iteration to update the player's movement. This update of the movement is carried out through a process called "update" which redefines the place of representation of Pac-Man depending on the direction of his movement.

To make Pac-Man's movement less complex for the user, what is called a “sustained movement” is implemented where, by means of a Player class shadow, it is evaluated, in case of a change of direction, if it is possible to perform change immediately, otherwise the last movement is held until the change is possible.

This reduces the precision requirement and the difficulty of controlling Pac-Man.

Because there are numerous objects, which must be created, we find that several of these objects receive from three to seven parameters, which in some cases could make it difficult to create and read the developed code.

Through the design pattern, constructor, the creation of most of the objects necessary for the project is handled.

The screen surface, the maze, the players, the Pac-Dots, and other objects are created.

Because only a few parameters need to be passed in the initialization of the constructor class, we find that the number of parameters that are sent for the creation of an object drops to two or three at its maximum extent. Being, even, in some cases zero.

In the process of creating objects, there are some of them in which it would not be necessary to have more than one instance of them, nor of the variables related to the object, in such a way that, in order to ensure a single instance of these objects, the Singleton pattern is implemented, which would allow it.

The existence of duplicates of certain objects, such as the labyrinth and ghosts, would result in an unnecessary waste of memory and computational space for the program.

For the creation of the ghosts, we take advantage of inheritance in Python, so a Ghost class is created from which all ghosts inherit the movement functions, and the states in which they can be.

These states are:

Initial State: When the ghost is inside the original box.

Active State: In this state they are dedicated to chasing Pac-Man.

Weak State: State in which they can be eaten by Pac-Man when he consumes a Power Pellet.

Eaten State: It is the state when Pac-Man has eaten the ghost.

In the creation of the ghosts, the factory pattern is used, where the ghosts already created, Inky, Blinky, Clyde and Pinky, would be received.

As part of the aspects of the game and the behavior of the ghosts we have that when Pac-Man consumes some of the Power Pellets, the ghosts would react in such a way that they would modify their movement.

When the ghosts are in a weak state, they will escape from Pac-Man to avoid being eaten.

In the eaten state, they would return to the box to be regenerated.

In the active state, they chase after Pac-Man

Due to the nature of the event when Pac-Man eats a Power Pellet, the observer pattern can be used.

The moment Pac-Man eats a Power Pellet, the ghosts should receive notification so that they know how they should act and what state they should be in.

In the observer pattern, an observer is notified when an observed object changes its properties, in this case, the Power Pellet would be touched by Pac-Man, and it would be the observed object. The watchers would be the ghosts.

Referring specifically to the movement, a simple way to do it is using the Manhattan function, where it is dictated that the difference, in this case, between the ghost and Pac-Man has a value (distance on the X axis added to the distance on the Axis y).

The ghost's movement consists of moving to a position that diminishes its Manhattan function. However, it would not always be possible to perform the movement that reduces this function as much as possible, so we always consider that four movements can be performed, up, down, left or right.

A list is made in which these four options would be ordered from the movement that reduces the Manhattan function the most to the movement that reduces it the least. The ghosts move to the first possible direction on this list.

Finally, the player's score is managed depending on what action he takes, whether he consumes a Pac-Dot, a Power Pellet or from one to all four ghosts. A class, called Score, would oversee making the sum relevant to the player's score through three different functions, one for each possible event.

Once all the functions, classes and patterns described have been developed, the recreation of Pacman using Python and the Pygame library is completed.

**3. CONCLUSIONS:**

The planning, development, coding and implementation of a project of an object-oriented paradigm must be clear, with a design where the classes and functions declared, together with the necessary objects, are expressed, where their interconnection is expressed. In the process of developing a project under these conditions, it is likely to encounter repeated problems where its solution can be found in the design patterns, which provide a useful template for solving these problems.

The design patterns used must be adjusted by the project developers so that they can fulfill their purpose as efficiently as possible; you can also take advantage of object inheritance.

In conclusion, a clear and efficient design has great repercussion

ns on the development and execution of projects.

**4. REFERENCES:**

[1] Source Making, *Design Patterns.*

Available:

https://sourcemaking.com/design\_patterns

[2] Pygame v1.9.2 documentation. (2017) .

Available:

https://www.pygame.org/docs