TRAIL OF TERROR

Ashwini Balasubramanian
Graduate Student
Clemson University
balasu6@clemson.edu

Adithya Ravi
Graduate Student
Clemson University
aravi@clemson.edu

Anjana Sriram
Graduate Student
Clemson University
anjanas@clemson.edu

Srivatsa Kandalam
Graduate Student
Clemson University
srivatk@clemson.edu

ABSTRACT

Trail of Terror is a Halloween-themed "escape-the-maze" game built in Python using the Turtle package. The goal of this game is to walk through a maze packed with ghosts and doors, designed to prevent the user from completing the game. The maze also contains two types of objects - crosses and keys - that, when gathered, helps the user to escape the maze. In this game, we are using the Singleton and Factory method design patterns. The goal of this game is to display our knowledge of the python programming language as well as to put into use the concept of design patterns in a useful and functional way.

Categories and Subject Descriptors

D.1.1: Applicative (Functional) Programming; D.1.5 [Programming Techniques]: Object-oriented Programming; D.2.10: Design; I.3.3 [Computer Graphics]: Picture/Image Generation; K.3.2 [Computers and Education]: Computer and Information Science Education

General Terms

Design

Keywords

Python, Singleton, Factory, Design Patterns, Object Oriented Programming, Turtle, Software Design, Game Development.

2. INTRODUCTION

We are creating a Halloween-themed maze game in Python utilizing two design patterns: Singleton and Factory. For the GUI, we use the Turtle library in python. Turtle is a Python library that comes pre-installed and is used to generate graphics, images, and games. The turtle library includes all of the methods and functions required for design and also the turtle graphics module is developed on top of tkinter, it is a strong library. We don't need to install any additional packages and only need to import the turtle package.

This is a simple and entertaining game. The player's goal is to effectively traverse the maze. To keep them from fleeing the maze, there are ghosts and doors. En route, the player must gather keys and crosses. A cross is required to assist in defeating a ghost, while a key is required to assist in opening a door. When the player makes contact with the ghost, the number of crosses decreases by one, and if the player does not have any crosses when the ghost approaches, the player loses a life. The player has three lives, after which he loses the game if he dies. The player wins the game if he or she successfully navigates the maze.

The goal of creating this game is to demonstrate our understanding of the python language and design patterns. Let us look into our program in depth and about all the methods and functions used to create this game in further sections.

2. CODE LAYOUT

2.1 Player Class

The first class we built in our program is the Player class, which is a singleton class and it is derived from the turtle class. Inside the constructor of this class we use shape for giving the player texture, penup for moving the player without leaving traces on the screen, speed for setting the animation speed to the quickest value, boolean variables key and cross that store if the player has a cross and a key respectively or not and an integer variable cross_count that stores the number of holy crosses. In this class, we have the following methods - go_up() is a function that moves the player up by one block, go_down() is a function that moves the player down by one block, go_right() is a function that moves the player right by one block, go_left() is a function that moves the player left by one block, and finally is_collision() is a function that determines whether the player's distance from a map element is less than one block.

2.2 Enemy Class

Class Enemy is our second class. This class inherits the turtle class as well. We have data members for a few tasks within the constructor of this class just like in player class namely, shape for giving the enemy texture, penup for ensuring the enemy does not leave any traces while moving, speed for setting the animation speed to the quickest value, direction to randomly choose a direction and goto for sending the enemy to its start point. The following methods are also available in this class. The move() method advances the enemy one block in a random direction, and if the player is nearby, the enemy stops moving randomly and begins chasing the player. If the next block is a wall, a cross or an unopened door, the enemy will move direction. This method is called repeatedly with a random delay until the game is over. The is_close() method was built to determine whether the player is close within a specific radius from the enemy. The last method is destroy(), which destroys the enemy.

2.3 mainCross, cross and crossNumber Class

Next, we have created the classes mainCross and cross. mainCross stores variables such as shape, speed and number of crosses that describe the crosses inside the game. As the player collects the crosses, we keep a count of the crosses that the player has. The goto() method is invoked from the Enemy class to send an enemy to the cross's location. destroy() method is used to destroy the enemies after the player-enemy contact happens when the player has the holy cross.The class cross is another class that is used to display or hide the image of the crosses on top of the screen if the player has or does not have the cross. The class crossNumber is used to increase or decrease the cross count. In the initializer, we use the turtle inbuilt methods shape, penup and speed to give the arrow a texture, render it with no trace and and set the animation value to the quickest value respectively.

2.4 Class Key and Key_Notif

Class Key creates objects for the keys inside the game. It stores the variables shape, prenup and speed for the key as in the previous classes. Variable key stores whether or not the key is present or not. Destroy() function is used to both move the key outside of the map and hide it from view. Key_Notif class is used to display the key symbol on top of the game screen if the player owns a key.

2.5 General purpose classes

The Pen class comes next and it, like the other classes, inherits the turtle class. The walls of the maze (pumpkins) are drawn using this class. The class BlackSquare is used to draw black squares for the game. These black squares are used to hide unnecessary items from the game window. Next we have the Health class which keeps track of the players health. This class has a destroy method to destroy a health symbol when the player is killed by the ghost. This class also inherits the turtle class. The next class is the Door class which is used to create doors in the game. This class has a destroy method, which destroys the door when the player opens the door with the key. Lastly we have the GameEnd, GameWin and GameOver classes. The factory design pattern is implemented here where GameEnd is the abstract class. These classes are used to determine if the player lost or won and to display the final message after the game ends.

2.6 The main loop

The part of the program loops till the end of the game. In the main loop, we first instantiate objects for each class and create lists to store various game items and their locations. Next we have a method called setup_maze() that sets up the GUI based on the characters it gets after reading a level from a list of strings. Each character in the LEVEL_1 list represents one item, for example, E represents enemy, P represents player, H represents heart, K represents key etc.
We use onekeypress to associate each keystroke with one movement of the player(up, down, left, right). Inside this loop, we display the crosses and keys on the screen as it is updated. We check to see if the player has a key every time he comes near a door and if he does we destroy that door. Similarly we check if the player has a cross everytime he comes near a ghost and if not he loses a life. We continuously check if the player has any lives left and if he doesn't it is game over. When a player collides with the exit block, the GameWin class is triggered and the game ends.

3. DESIGN PATTERNS

3.1 Singleton Design Pattern

Singleton design pattern is a creational design pattern in which only one instance of a class can be created. There are many benefits in creating a singleton class, some of which are that it prohibits many users from using a shared resource at once, it is useful in creating a global access for a shared resource and it is helpful when the program needs only one instance for its entire lifetime. In our program, we are using the singleton design pattern to create the class Player as we need only one instance of the player class to be created.

```
class Player(turtle.Turtle):
    def __new__(cls):
        if not hasattr(cls, 'instance')
            cls.instance = super(Player
        return cls.instance

    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape(r"player_right.gif")
        self.penup()
        self.speed(0)
```

Fig 1. Class Player in Singleton design pattern

In this code, the __new__ method will check whether an instance is created or not. If created, If the instance already exists, it will return it; otherwise, a new instance will be created.

3.2 Factory method Design Pattern

Factory is also a creational design pattern which provides an interface to create objects. This design pattern lets the subclasses decide on the type of object that is created for the superclass. Factory is particularly useful when the class does not know the type of subclasses in advance or when it wants the subclasses to choose the object to be created.In our program we are using the factory design pattern in the GameEnd class.

```
class GameEnd(turtle.Turtle):

    @abstractmethod
    def __init__(self):
        pass

class GameOver(GameEnd):
    def __init__(self):
        turtle.Turtle.__init__(self)
        self.shape("square")
        self.color("red")
        self.penup()
        self.speed(0)


class GameWin(GameEnd):
    def __init__(self, x, y):
        turtle.Turtle.__init__(self)
        self.shape(r"exit.gif")
        self.penup()
        self.speed(0)
        self.goto(x, y)
```

Fig 2. GameEnd implements Factory design pattern

In this program, GameOver and GameWin classes should implement the GameEnd Interface. These classes can implement the interface differently. The GameWin interface is an abstract class.

4.Output

The output of this game is shown below.

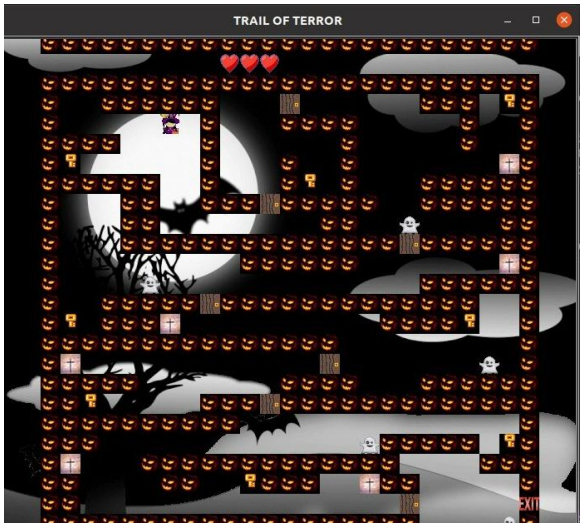

Fig 3. Screenshot of game main screen

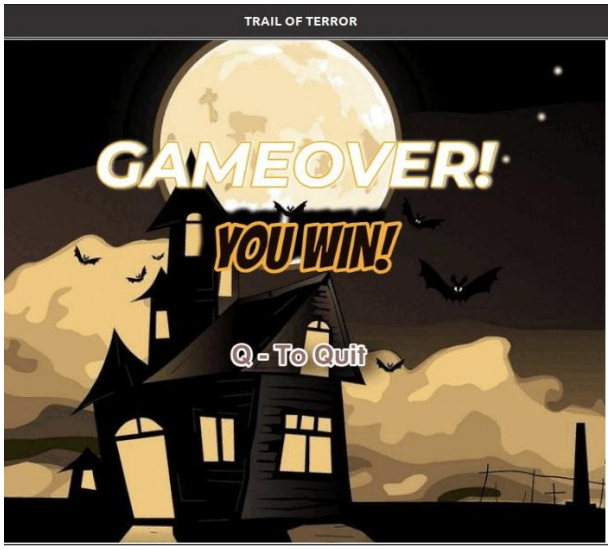

Fig 4. Screenshot of game screen after winning

Fig 5. Screenshot of game screen after Loosing

5. CONCLUSION

We have successfully implemented a simple halloween themed game using python and the turtle package. Our code has thus displayed the use of two design patterns with many functionalities in it. There is room for improvement to make this game more enjoyable for commercial purposes.

5. REFERENCES

1·gif·: Motion Design Animation, Motion Graphics Design, powerpoint background design (2015) *Pinterest*. Available at: https://in.pinterest.com/pin/462252349229094203/ (Accessed: November 29, 2022).

Anonymous (2015) *Cross GIF*, *Gfycat*. Available at: https://gfycat.com/courageouselementaryhalicore (Accessed: November 29, 2022).

Design patterns in python (no date) *Refactoring.Guru*. Available at: https://refactoring.guru/design-patterns/python (Accessed: November 29, 2022).

Factory method - python design patterns (2022) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/factory-method-python-design-patterns/ (Accessed: November 29, 2022).

Funimada.com (2022) *Original GIF images*, *Funimada*. Available at: https://www.funimada.com/ (Accessed: November 29, 2022).

Goldwasser, M.H. and Letscher, D. (2008) "A python graphics package for the first day and beyond," *ACM SIGCSE Bulletin*, 40(3), pp. 326–326. Available at: https://doi.org/10.1145/1597849.1384369.

Halloween (2022) *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Halloween (Accessed: November 29, 2022).

Mine, M.R., Shochet, J. and Hughston, R. (2003) "Building a massively multiplayer game for the million," *Computers in Entertainment*, 1(1), pp. 1–20. Available at: https://doi.org/10.1145/950566.950589.

Purnamasari, O. (2020) *Download happy Halloween from the Spooky Castle for free*, *Vecteezy*. vecteezy. Available at: https://www.vecteezy.com/vector-art/1331268-happy-halloween-from-the-spooky-castle (Accessed: November 29, 2022).

Python design patterns - javatpoint (no date) *www.javatpoint.com*. Available at: https://www.javatpoint.com/python-design-pattern (Accessed: November 29, 2022).

Python turtle programming tutorial - javatpoint (no date) *www.javatpoint.com*. Available at: https://www.javatpoint.com/python-turtle-programming (Accessed: November 29, 2022).

Turtle programming in python (2021) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/turtle-programming-python/ (Accessed: November 29, 2022).

(no date) *Google image result for https://wallpaperaccess.com/full/1212170.jpg*. Google. Available at: https://www.google.com/imgres?imgurl=https%3A%2F%2Fwallpaperaccess.com%2Ffull%2F1212170.jpg&imgrefurl=http%3A%2F%2Fpattayathailand.ru%2Fgo%2Furl%3Dhttp%3A%2Fagreen.top%2Fhalloween-animated-gif-wallpaper&tbnid=ZXvr6FfLifBRqM&vet=1&docid=wKuJUoZgyPAGaM&w=1600&h=1000&itg=1&source=sh%2Fx%2Fim (Accessed: November 29, 2022).