

**REPORT**



**OBJECT – ORIENTED PROGRAMMING**

**Plants vs. Zombies**

Group member:

* Nguyễn Thành Công – ITITIU19088
* Trần Cao Bảo Phúc – ITITIU19
* Nguyễn Hà Minh Trí – ITITIU19
* Võ Vương Nhật Tân – ITITIU19

*Score: \_\_\_\_\_\_\_\_\_*

**

**CONTENTS**

1. **Introduction** 3
2. **Goal** 3
3. **Basic Game Rule** 3
4. **Preparation** 3
5. **Class** 3
6. **Object** 4
7. **Constructor** 4
8. **Error catching** 5
9. **Logic Algorithm and Function** 5

* **Class Logic** 5

1. **Board Game** 5
2. **Load Image** 6
3. **Mouse Interaction** 6

* **Class Plant** 7

1. **Draw** 7
2. **Shoot** 7

* **Class Zombie** 8

1. **Move** 8
2. **Attack and Collide** 8
3. **Methodlogy** 9
4. **Result – Limit – Conclusion**  11
5. **INTRODUCTION**

It is the speed up rapidlly of the Information Technology industry, the need of advanced technical skill is always required. It is because of the classic *Procedural – Oriented Programming* method cannot maintain the development of this programming side, this leads to the appearance of a new method *Object – Oriented Programming*.

This project is based on the Object – Oriented Programming method by using Python language

Plants vs. Zombies is a tower defense and strategy video game developed and originally published by PopCap Games for Windows and OS X in May 2009, and ported to consoles, handhelds, mobile devices, and remastered versions for personal computers.

1. **Goal:**

The aim is simple. You collect and save the “SunCoin” which fall down from the top of the game display or from the Sun Flower:

- To defend their home from zombies, some of which have unique abilities, the player uses plants that can fire projectiles at, or have other detrimental effects on, the horde of advancing zombies.

- The player collects "sun" to buy defending plants.

- Zombies approach along several parallel lanes on the lawn, and the player must plant defenses in these lanes. If a zombie get to the house on any lane (total 5 lanes), the game is over.

1. **Basic Game Rules:**

* Planting to defeat all the zombies from each level to get the victory, if one of them reach to the end of the lanes (left side), you will fail.

1. **PREPARATION**
2. **Class:**

In Object – Oriented Programming, A class is a code template for creating objects. Objects have member variables and have behaviour associated with them. In python a class is created by the keyword class.

The user – defined objects are created using class keyword. The class is a blueprint that defines a nature of a future object. An instance is a specific object created from a particular class. Classes are used to create and manage new objects and support inheritance – a key ingredient in object – oriented programming and a mechanism of resuing code.

\*\*There are 2 main class: Zombie and Plant that we want to concentrate on because we are applying the Object – Oriented Programming method to this project.

1. **Object:**

An object is created using the constructor of the class. This object will then be called the instance of the class. In Python we create instances in the following manner

Object determines the behavior of the class. When you send a message to an object, you are asking the object to invoke or excute one of its method. From a programming point of view, an object can be a data structure, a variable or a function. It has memory location allocated. The object is designed as class hierarchies.

1. **Constructor:**

Each time an object is created a method is called. That methods is named the constructor.

The constructor is created with the function ***init***. As parameter we write the self keyword, which refers to itself (the object). For example,

|  |
| --- |
| class Human:  def \_\_init\_\_(self):  self.legs = 2  self.arms = 2  bob = Human()  print(bob.legs) |

1. **Error Catching:**

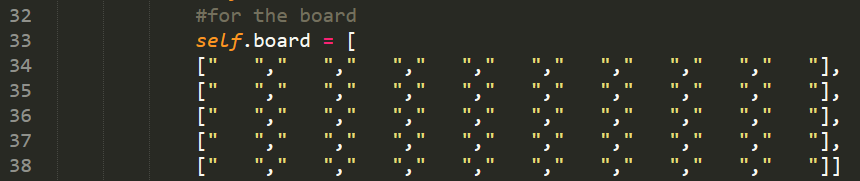
|  |
| --- |
| Try:  #Your code here#  except Exception as Bug:  print("game.py:",Bug)  print("Please report the bug")  Input("Double Press Enter to continue !!")  input() |

1. **Logic Algorithm**

* **Class Logic:**

1. **Board Game:**

We have a big class Logic. In this class, we have the constructor which contains the board of the game:



And other method to support the board game: setBoard() and getBoard()

1. **Load Image:**

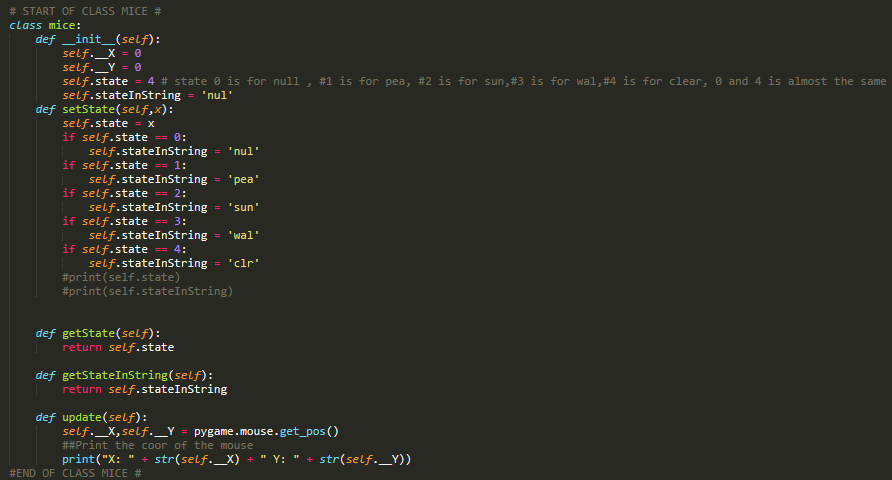
|  |
| --- |
| **ImgList** = []  def **loadResource**(ImgList):  num = 1  while(num <= 30):  ImgList.**append**(pygame.image.load('Resources/blank.png'))  **num** = num + 1  return ImgList  **plantImgList** = **loadResource**(plantImgList) |

**\*\* Explain:**

* The movement pattern of plant and zombie is 30 different images so the variable "**num**" ends at 30.
* The variable “**ImgList**”: Creates a motion picture list of plants and zombies.
* The **append()** method adds a single file to the existing list.

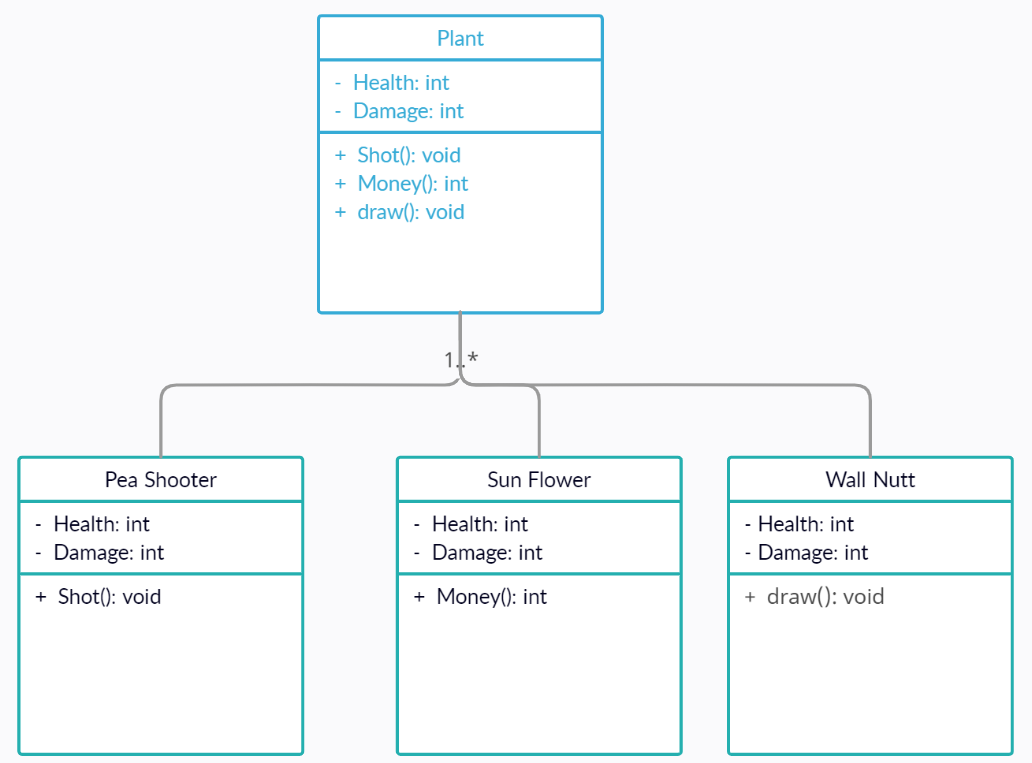
1. **Mouse Interaction:**

Mouse click is also one the most important things that we had added to this Logic class. We have named it Mice class

****

**\*\* Explain:** This is the one which show you the position of the mouse currently. When we try to add the new object (a new plant or clear a plant) by clicking into the place where we want to set, it will save the position values and fill the number of the plant (PeaShooter is 1, SunFlower is 2,…) in the state of the board

* **Class Plant:**

****

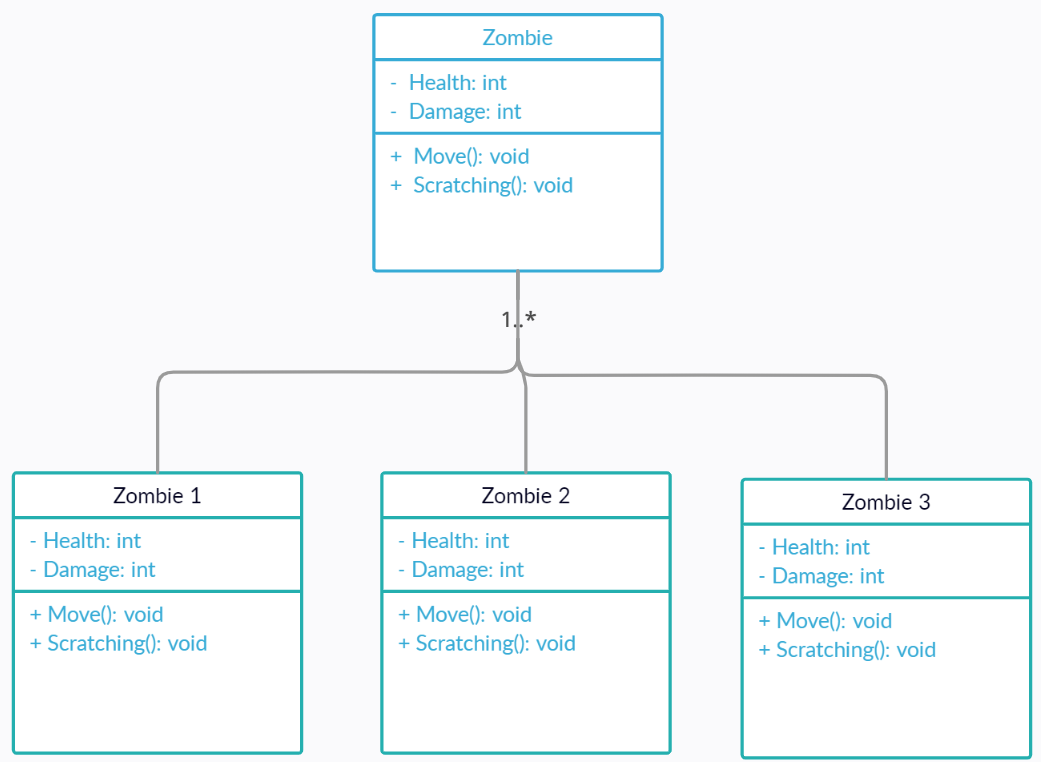
All the class PeaShooter, SunFlower and Wall Nutt inherit the father class Plant

1. **Draw:**

|  |
| --- |
| def **draw**(self,window):  if self.health <= 300 and self.health >= 200:  **self.\_\_frame = 0**  elif self.health < 200 and self.health >= 100:  **self.\_\_frame = 1**  else:  **self.\_\_frame = 2**  **window.blit(self.ImgList[self.\_\_frame],(self.x,self.y))** |

**\*\* Explain:**When the blood of a Plant or Zombie decreases to a certain level, the appearance will change.

* **Class Zombie:**

****

1. **Move:**

|  |
| --- |
| def **move**(self):  if self.x >= 330:  self.x = self.x - 1 |

**\*\* Explain:** Zombies move from right to left, so every second Zombie moves, X coordinates will decrease by 1 unit to the end of the map's moving range.

1. **Attack & Collide:**

|  |
| --- |
| def **attack**(self,inPlant):  inPlant.health = inPlant.health - 2.5  if(inPlant.health <= 0):  self.stop = 0 |

|  |
| --- |
| def **isCollide**(self,inPlant):  if(self.x == inPlant.rightBorder and self.r == inPlant.r):  self.stop = 1  self.attack(inPlant) |

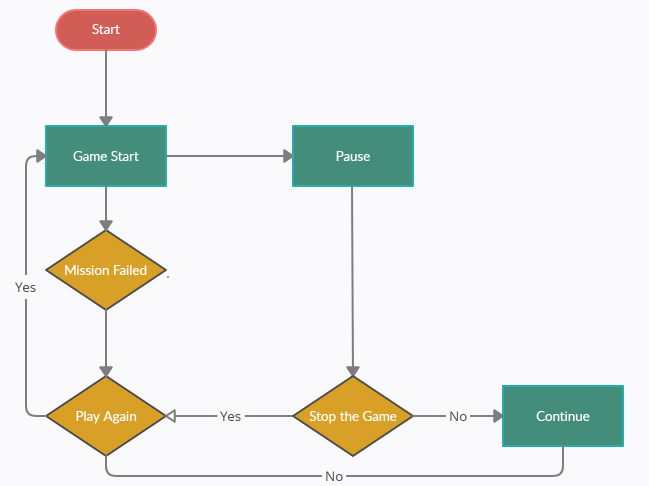
**\*\* Explain:** When a Zombie collides with a Plant, "isCollide" will be activated, "attack" will also be activated to subtract the Plant's Health.

1. **Methodlogy**

This game project has been made by combinating totally 5 python files: main.py, game.py, menu.py, plant.py and zombie.py

All the logic function of the game were added in the Logic class inside game.py. To run the program, we all set it in the main.py with some attributes and actions, events of the map which is delared in class Logic before.

**All the process of the game:**



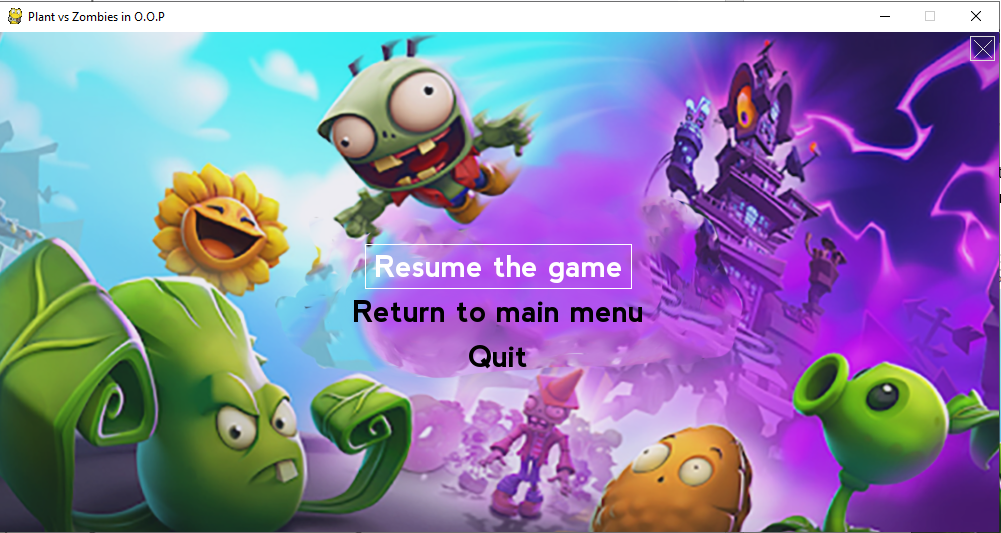
1. **Result – Limited – Conclusion**
2. **Result**

* Based on the principles of The OOP Method, our project comes complete with the basic Plants & Zombies rules and properties. The association of classes and objects in the system is relatively logical. We have successfully implemented the same user interface requirements as the original Plants & Zombies version such as Main Menu, Pause Menu, Sun Money. **Moreover**, the animation of plants and zombies is smooth, **Sorry, tiếng anh của t quá tệ**
* **Main Menu:**

Graphical user interface

Description automatically generated

* **Pause Menu:**



* In Game:

**A screenshot of a video game

Description automatically generated**

1. **Limit**

Although the game is basically worked, we have some cases which cannot be solved:

* Do not have the sun falling down from the top of the game, player name and high score saving function.
* There is only one level, we have not update new level (easy, normal, hard) and add more kind of plants and zombie.
* The setting of sound is still considered, we are trying to fix the delay of bullets of Pea Shooter when there is no zombie.

1. **Conclusion**

"Plants and Zombies" was built by an object-oriented method. This shows clearly inheritance, polymorphism, encapsulation, data abstraction of OOP, and the relationship between Plants, and Zombies are linked tightly and systematically. Besides that, acquiring a lot of knowledge outside the course is one of the important things in the process of making "Plants & Zombie".