Assignment 3.1

CASTLE CLASH

Overview

- A 2D game in Python3 (terminal-based) using the concept of Object Oriented Programming, heavily inspired by Clash of clans where the user will control the king, move it up, down, forward and backward, while destroying buildings and fighting defences on it's way.
- The objective of the game is to destroy as many buildings as possible, and collect the maximum amount of loot while doing so. There will be an army of troops to help the king clean up.

Controls

- You can control the King's movement throughout the map using the keys **WASD**.
- You can make the King attack in the last moved direction using **SPACE**.
- You can use the King's leviathan axe for an Area of Effect attack using **Q**.
- You can spawn troops/barbarians at three different locations around the village using keys **1**, **2** and **3**.
- You can use the heal spell for your king and every alive troop using key **h**.
- You can use the rage spell for your king and every alive troop using the key **r**.
- You can press key **e** for a replay of all your attacks.
- You can use the key **0** to exit from the game

Description of Classes Created

Game:

The game class creates a 26*45 grid for gameplay with boundaries, walls, empty space, buildings(townhall, canon and huts), king and troops. The draw_grid() method prints this grid on the terminal.

Building:

The Building class is the base class based on which all other buildings of the village are inherited.

Townhall:

The townhall class is inherited from Building class and has additional functionality.

Hut:

The Hut class is inherited from Building class and has additional functionality.

Canon:

The Canon class is inherited from Building class and has additional functionality of attacking the enemies(king/troops) and some additional private data members like range and damage.

Wall:

The Wall class is inherited from Building class and has additional functionality. It also represents polymorphism as the render method() is changes/overridden.

Enemy:

The Enemy class is the base class based on which king and troop class are inherited.

King:

The king class is inherited from Enemy class and has some additional functionality like controlled movement, controlled attack and range attack.

Troop:

The troop class is inherited from Enemy class and has some additional functionality like automated movement and attack.

Spell:

The spell class is base class for the spells that the king and troops can use.

Rage:

The Rage class is inherited from Spell class and has additional functionality of increasing the damage and movement speed of the king and all alive troops.

Heal:

The Heal class is inherited from Spell class and has additional functionality of increasing the health of the king and all alive troops.

Concepts used

Inheritance:

Inheritance allows us to define a class that inherits all the methods and properties from another class. A base class `Building` has been declared from which multiple elements are inherited.

```
class Building:
    def __init__(self):
        # default
        self.destroyed = False
        self.health = 100
        self.width = 0
        self.height = 0
        self.char = ''
        self.start_row = 0 # default
        self.start_col = 0 # default
        self.type = 0 # 1 for wall

    def destroy(self, damage):
        self.health = self.health - damage
        if(self.health <= 0):
            self.destroyed = True

    def render(self, game, start_row, start_column):
        self.start_row = start_row
        self.start_col = start_column

        if (self.destroyed == False):
            for row in range(start_row, start_row + self.height):
                for column in range(start_column, start_column +
self.width):
                    game.update_tile(row, column, self.char, self.health,
self)
```

Polymorphism

Polymorphism allows us to define methods in the child class with the same name as defined in their parent class. eg.

```
class Building:
    def __init__(self):
        ...

    def render(self, game, start_row, start_column):
        self.start_row = start_row
        self.start_col = start_column

        if (self.destroyed == False):
            for row in range(start_row, start_row + self.height):
                for column in range(start_column, start_column +
self.width):
                    game.update_tile(row, column, self.char, self.health,
self)

class Wall(Building):
    def __init__(self, row, col):
        ...

    def render(self, game):
        if (self.destroyed == False):
            game.update_wall_tile(self.row, self.col, self.char, self)
```

Encapsulation

The idea of wrapping data and the methods that work on data within one unit. Prevents accidental modification of data. Implemented many classes and objects for the same.

Abstraction

Abstraction means hiding the complexity and only showing the essential features of the object.

```
class King(Enemy):
    # user-controlled character capable of attacking and destroying
buildings
    def __init__(self, curr_row, curr_col):
        ...

    def move(self, game, c):

        updated_row = self.curr_row
        updated_col = self.curr_col

        # w
        if(c == 1):
            updated_row = self.curr_row - self.speed
            self.last_move = 'w'
            for row in range(self.curr_row - self.speed, self.curr_row):
                if(row >= 0 and row <= rows - 1):
                    if(game.grid[row][self.curr_col] != Back.CYAN + "." +
Style.RESET_ALL):
                        return
        ...
```

.move() is an abstraction

How To Play:

- Download the code from the github repository and run it in the terminal, **python3 main.py**.

Requirements:

- Python3