CS6.301: Design and Analysis of Software Systems Spring 2020

Assignment-3.1

Released: February 19th Deadline: March 5th

Instructions

- This assignment is designed to get you familiar with object oriented programming (OOPS) concepts. You have to use classes and objects to write good and modular code.
- Submit all your code files and a README.md as a zip file, named as ⟨rollnumber⟩.zip.
- The readme should describe your game, its rules, all the functionalities and any cool features! It must act like a quick start guide, describing all the controls and features.
- The deadline is March 5th, 23:55. This is a hard deadline and no submissions will be accepted after this.
- Start early! This assignment may take fairly long.
- TAs to contact : Bhavyajeet, Koushik, Abhijeeth, Pavani, Alapan
- Plagiarism detectors will be run on all submissions, so please do not copy. If found, you would be given a straight zero for the assignment.
- Please write a **modular** and **extensible** code for this assignment.

Introduction

Ashish is the founder of a clan called proraiders and wants to recruit new members to his clan. However he is very picky and wants all his new recruits to first show their skills at a miniature version of the clash of clans game. Help him by implementing this game so that he can use the same for testing his recruits.

For this assignment, you need to build a 2 D game in Python3 (terminal-based), 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 its way. Concepts of object oriented programming must be present within your code and the game must simulate a basic version of Clash of clans .

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.

OOPS Concepts (20):

Your code must exhibit OOPS concepts, that is the main objective of the assignment.

- Inheritance: You could have one building class and have the different kinds of building inherit from the building class.
- Polymorphism: You could have one spell class and override various characteristics to exhibit different properties.
- Encapsulation: The fact that you're using classes and objects should suffice for this! Use a class and object based approach for all the functionality that you implement.
- Abstraction: Intuitive functionality like move(), attack() etc should be methods within the class. So for instance (no pun intended), if you declare an object 'player' of the person class, you can call the methods player.move() and player.attack(), stowing away inner details from the end user.

Game Specifications:

Village (20):

There will be a map of n*m dimensions representing your village level. Each village must have the following properties

- Spawning points:
  - There should be **three** predefined spawning points around the village.
  - Each spawning point will be controlled by a different key. Pressing the key of a certain spawning point will cause a troop to be spawned there.
- Town Hall:
  - The Town Hall is the central building of your village.
  - There is only **one** town hall per village.
  - Size: 4x3 tiles
- Huts:
  - There should be at least **five** huts in your village
  - Size of the hut can be defined by you.
- Walls:
  - There should be a sufficient number of walls in your village to protect your town hall from troops.
  - Size: 1x1 tiles
- At least two cannon
- Cannon:
  - There should be at least **two** cannons in your village.
  - Size can be defined by you.
  - The cannon will have a range and damage value (can be defined by you). The range (must be greater than 5 tiles) would define the area till which it can attack and the damage value should be a number telling the amount of damage it yields to a single troop in a second.
  - At a given point, the cannon can only target a single troop.
- Each building will have a certain amount of hitpoints (basically health) and the remaining hitpoints of the building would be denoted by its colour.
  - The hitpoints of the building should be split into at least **three** ranges, each with a different colour.
    - For example: Green: 50% to 100% hitpoints, Yellow: 20%-50% hitpoints, Red: 0%-20% hitpoints.
  - A building with 0 hit points is considered *destroyed* and should not be displayed on the screen or targeted by troops.

King (20):

The king is a user-controlled character capable of attacking and destroying buildings.

- Controls and Movements:
  - The King is to be controlled with W/A/S/D which correspond to Up/Left/Down?Right.
    - The King's movement speed is left to you to decide
  - <SPACE> will be used for attacks
    - The King will attack a single location with a sword. This location should be specified relative to the location of the king, any building present in that location would be damaged by the king's attack.
- Attributes:

  The values of each of the following attributes can be defined as you wish, the constraints on the values are given below.

  - Damage: The damage each attack of the king deals to a building
    - The damage the king deals should NOT exceed the maximum health of ANY building (i.e. the King can't one-shot anything).
  - Health: The king's health should be displayed as a health bar anywhere on the screen.
    - The health of the king should be higher than the damage dealt by any defensive building.
    - When the health of the king drops to 0, the King would be dead and cannot

move or attack anymore.
    - Movement Speed: The distance that the king moves in each time step.

Barbarians (15):

- Each barbarian will have the following attributes
    - Damage - the amount of damage it will yield per attack
        - Every troop attacks once per time step
    - Health - the hitpoints it has - The health of the troop would be depicted by the colour of barbarian … the colour should change from dark to light with a change in health. This shift can be discrete.
        - Once the health of a barbarian drops to 0, they are considered dead and cannot move or attack any longer.
    - Movement speed

- Movement : The barbarians will always try to attack the nearest non-wall building and will always move towards it. The barbarian movement is automated. If there is a wall in its path then the barbarian is expected to first destroy the wall and then move forward.

Spells (5):

- Rage Spell:
    - The Rage spell affects every troop alive in the game and the King.
    - It doubles damage and movement speed.
- Heal Spell:
    - The Heal spell affects every troop alive in the game and the King.
    - It increases their health to 150% of the current health (capped at the maximum health)

Game Endings (5):

- Each game can end in either victory or defeat.
    - Victory: All buildings (excluding walls) have been destroyed.
    - Defeat: All troops and the King have died without destroying all buildings.
- Once either of these conditions is satisfied, the game would end.
- You are required to display the result of the game once it ends.

Replay (15):

- Implement a replay feature for all attacks. Any game that has been played should be available as a replay.

Bonus

- King's leviathan axe: the king instead of attacking a single building with a sword, now uses an axe to do an AoE (Area of Effect) attack to all buildings in a specific vicinity. (10 marks)
    - For example: The King when using this attack would deal damage to any building in a 5 tile radius around him.
- Extra marks might be awarded for any other creative feature. (5 marks)
    - You are free to add any features present in the original game or come up with new features of your own.
    - Ensure that the feature uses different logic from what you have implemented in other parts of the game
    - Ensure that the feature does not override or erase any other feature in the game.

Library Usage

• Your game must mandatorily be coded in Python3.

• No curses libraries (like pygame) are allowed. Only libraries allowed are colorama and numpy. In case of any doubts about whether a particular library is allowed, please post it on the Moodle

thread and get it clarified from one of the TAs.

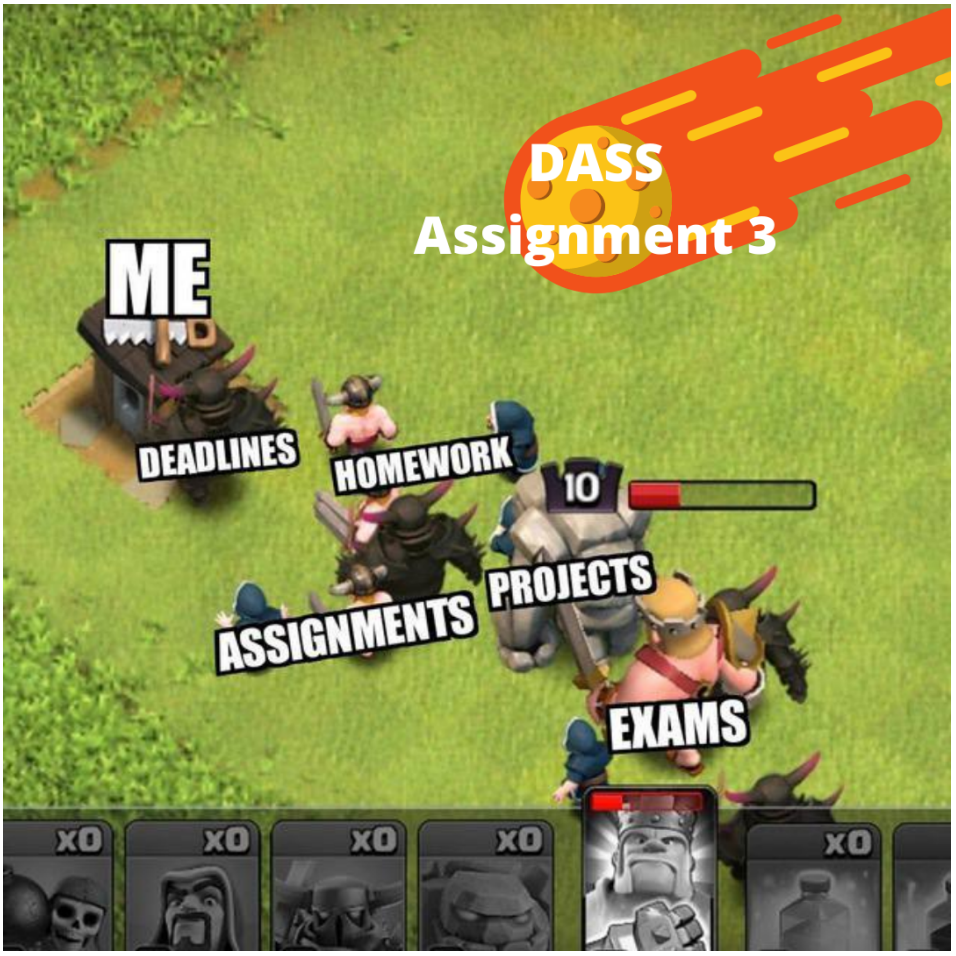
Deliverable

File Structure:

- <Roll Number>
  - README.md
  - game.py: The main game file which will be run to play the game
  - src: folder containing any other code files
  - replays : folder storing the necessary files for the replay
  - replay.py : Python file needed for rendering a replay

A zip file, ⟨rollnumber⟩.zip containing the code files and README. For instance, 2018102022.zip when extracted must generate a folder with the name 2018102022. Inside that folder, all code files must be present. Ensure your code is modular with multiple python files. Please ensure to follow the submission format and submit ONLY .zip files.

Plagiarism is a strict NO, and Koushik is running moss 😈 So please do not copy from anyone. We'd be forced to give you a zero for the assignment
It is your game, make it as creative and fun as you can!



Happy coding 😇