Basic Information

Team Number 42

Team Members

Name	Roll Number
Sreeram Reddy Vennam	2021101045
Akshat Sanghavi	2021101094
Keshav Gupta	2021101018
Haran Raajesh	2021101005

Contributions

Name	Contribution
Sreeram Reddy Vennam	UML Diagram, found bugs and code smells
Akshat Sanghavi	Table Summarizing Classes, found bugs
Keshav Gupta	found code smells and bugs
Haran Raajesh	found code smells and bugs, implemented refactoring for 3 code smells

Overview

The software system studied is a CLI game written in python inspired by Clash Of Clans.

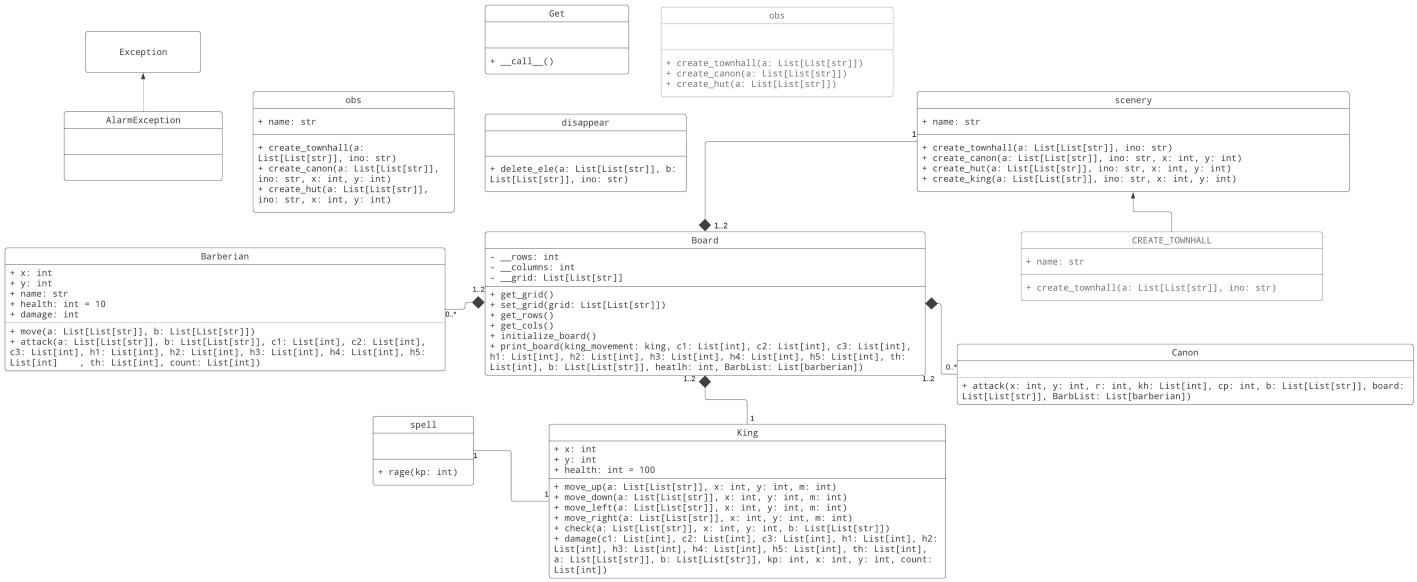
In this game, you play as a king, commanding barbarians to destroy cannons and huts and the townhall all within the confines of a village. The king can lose damage from cannons and thus you are allowed to use a heal spell to regain health. You can also use a rage spell to increase damage and move faster. The game updates state every 2 seconds or whenever a key is clicked.

Features

- Colors represents health of building
- King, can move and attack buildings
- Rage and heal spells that modify king behaviour
- Replays, allows replaying games played
- Canons attack barberians and the king
- Barberians can be spawned and move towards huts
- King can attack and cause AOE damage

Class Structure

Click https://iili.io/Hh72SZg.png for a clear image.



Class Descriptions

Class Name	Responsibility
Board	Keeps state of the game, i.e. where the pieces are placed inside the village.
Scenery	Helper class that contains useful functions to create static elements of the village like the hut.
Canon	The canon class stores information about the canon and contains a function that applies cannon damage.

Class Name	Responsibility
King	The class that implements the king, has functions to move the king on the board and attack buildings.
Barberian	The class that implements the barberian, has functions to move and attack.
Spell	The class that implements the rage spell.

Code Smells

The refactoring column is optional, `na` if not done

Smell	Description	Refactoring
Long Parameter List	In `barberian.py`, the `barberian` class has the method `attack` which has 13 parameters	We can combine parameters `h1` to `h5` into a list `h: List[]` and parameters `c1` to `c3` to a list `c: List[]`
Long Parameter List	Similarly in `king.py` also, the `king` class has the method `damage` which has 15 parameters	We can combine parameters `h1` to `h5` into a list `h: List[]` and parameters `c1` to `c3` to a list `c: List[]`
Long Parameter List	In `board.py`, the `board` class has the method `print_board` which has 14 parameters	We can combine parameters `h1` to `h5` into a list `h: List[]` and parameters `c1` to `c3` to a list `c: List[]`. We can also remove `king_movement`, `health`, and `BarbList` parameters as they haven't been used in the function
Duplicated Code	In `barberian.py`, the `barberian` class has the method `attack` which has a large if-else if ladder where each block carries similar functionality of causing damage to a particular object and removing the object, if its health is less than or equal to zero	If the above refactoring suggestion is followed, we have arrays, for the huts and cannons, so we can use simple string matching to reduce the amount of conditional blocks required

Smell	Description	Refactoring
Duplicated Code	In `king.py`, the king class has 4 differnt methods for just one type of motion, `move_up`, `move_down`, `move_left`, and `move_right`	We can have a single method `move` which takes in the direction as a parameter and then moves the king in that direction
Uncommunicative Name	In `barberian.py`, the `barberian` class has the method `move` which has parameters `a` and `b`. These variables are grids but this information is not conveyed anywhere in the class nor in the variable names.	Rename `a` to `board_grid` and `b` to `obs_grid`.
Refused Bequest	In `object.py`, the class `CREATE_TOWNHALL` inherits from class `scenery` but never uses any parent functions, in fact, the method `create_townhall` is identical the method `create_townhall` in the parent class `scenery`.	Remove the use of inheritance
Oddball Solution	In `canon.py`, class `canon` is used to represent all classes, the method `attack` loops over every canon and attacks for all of them.	The `canon` class should represent **1** canon and multiple instances of this class should be instantiated for multiple canons.
Duplicated Code	In `board.py`, the class `board` has a method `print_board`, which has a large if-elif ladder, where groups of the blocks execute the same code	Could concatenate the groups where the same code is executed into once condition, and thus have only one conditional block for each unique code block
Dead Code	In `board.py`, the class `board` has a method `print_board`, which has 14 parameters, 3 of which are unused	Removal of the unused parameters
Long Parameter List	In `board.py`, the `board` class has the method `print_board` which has 14 parameters	We can combine parameters `h1` to `h5` into a list `h: List[]` and parameters `c1` to `c3` to a list `c: List[]`

Smell	Description	Refactoring
Dead Code	In `canon.py`, the class `canon` has a method `attack`, which has 8 parameters, 2 of which are unused	Removal of the unused parameters
Long Method	In `canon.py`, the class `canon` has a method `attack`, in which the developer has written code thaat contains 2 for loops that iteratoe over every grid-cell in the range of each cannon, to check if a the King or a Barbarian is in that grid-cell or not	We can just check if the King and the Barbarians are in range by using the distance formula instead pf checking each grid-cell
Data Class	In `king.py` the king class contains methods that doesn't use the data of the class, but rather uses arguments which represent the same values	Methods can use less arguments and instead use the data of the class
Data Class	In `object.py`, `scenery`, `CREATE_TOWNHALL`, and `obs` classes don't use the data stored in the class, all methods in these classes modify external variables. They set but do not use any class variables.	Methods need not be placed inside a class. Instead, they should be part of a class that modifies the data that is passed as parameters.
Data Class	In `king.py` the king class contains methods that doesn't use the data pf the class, but rather uses arguments which represent the same values	Methods can use less arguments and instead use the data of the class
Duplicated Code	In `play_game.py`, the developer has rewritten the same piece of code in the if-else conditional block (in the game loop), when the space input (' ') is being conditioned, only change in the different cases is the name of the object variable (cannon, huts etc).	We can create a common function and take the object variable as the argument, and just call that function from the condition block
Duplicated Code	In `king.py`, again the developer is using almost the same code in multiple if-else blocks, with only	Creating a common method and using that in the conditioning, with the object variable as the

Smell	Description	Refactoring
	change being the variable accessed, which would instead suit better as a function argument	argument to that method
Dead Code	In `barberian.py` the constructor method needlessly has the observer-grid as an argument	Remove that argument from the init method
Dead Code	The file `observer.py` is never used, it contains a class `obs` but it was redefined in `object.py` and `obs` is always imported from `object.py`.	Deleting the file `observer.py`.
Data Class	In `spell.py`, class `spell` does not contain any data, only a function that works only with the parameters passed.	Method need not be in a class.
Oddball Solution	In `spell.py`, class `spell` implements the rage spell but the heal spell is not implemented.	Implement the heal spell as a class inheriting from the `spell` class
Lack of Documentation	No file except `play_game.py` has any explanatory comments	Write comments for hard to understand to code.

Bugs

Bug	Description	Refactoring
No Win Condition	In `play_game.py:102:5`, to win the game we check if the `count` variable is `9`. However, `count` is a `List[]` and thus the game is never winnable.	The check should be `if (count[0] == 9)` but better yet, we never `push` to or `pop` from the `count` list and so it can simply be made an integer.
Canon attacks after death	In `play_game.py:87:26`, the canon damage is applied regardless of whether the canon is destroyed or not (the check isn't done in `canon.attack()` either).	First, we check if the canon is alive using canon health variables `c1` to `c3` and only attack if the health is non-

Bug	Description	Refactoring
		zero.
Barbarian moves toward the cannons	In `barberian.py`, the barberian moves towards one of the elements of `building_pos`, which in `config.py:12:5` also includes the cannon locations.	We can simply remove the cannon locations from `building_pos` and thus the barbarian will not move towards the cannons.
No consistent time step	The game refreshes after every key click or every 2 seconds (`timeout` in `input.py:32:21`) if no key is clicked within that time. The pdf seems to hint that the game should refresh in constant time steps and should be independent of what key the user clicks.	We can create a buffer that stores the last key pressed before the game resets, when the game resets we can check this buffer and consider what it contains as user input.