DASS Assignment - 4

Code Reviewing & Refactoring

TEAM : 43

NAME	Ayush Sharma	Rajat Kumar	Mc Bhavana	M Pravalika
ROLL NO.	2019101004	2019101020	2019101100	2019101098

Short Overview Of System : **The Mandalorian**

Since, no readme or report has been provided to us it was very difficult to understand the various functionality in the code & then reverse engineer it. We extracted following information about the system:-

- The software system given was of a CLI game using basic ASCII printing and re-rendering of the game frames.
- The system was written in Python and did not contain any explicit library.
- The codebase seems to be modular and is written using the basic principles of Object-Oriented Programming which covers the concepts like Inheritance, Abstraction, Polymorphism and Encapsulation.
- There is also a class that enables the user to provide inputs via the terminal and then displays the frames according to the action cause because of that input like ‘**a**’ to move left, ‘**w’** to move up ,**‘d**’ to move right ,**‘l’** to shoot, **‘q**’ to quit and **‘Spacebar**’ to activate shield.
- The game seems to be a one where there are enemies(obstacles) that the player has to avoid and can shoot objects, and collect various coins, shield(It can be activated to protect Mandalorian from obstacles), power ups which can last upto 5 secs and fight with fierce dragon to win the game which looks like it is modelled similar to terminal version of Jetpack Joyride.
- The design of the game seems to be object oriented and can be implemented mostly based on the creativity of the developer.
- The player may speed up for some time by hitting the power up obstacle.

Narrative Analysis Of Original Design

Strength

The original design was developed according to OOP principles. It consists of a class-based and object-based implementation, with functions to implement its attributes and characteristics, such as movement and interaction. In design, time is used as the basis for most operations. The infinite loop actively searches for collisions between the various components of the game. Strengths. The game is designed so that the player appears to be moving in an infinite world by sliding objects in and out of the game. The game appears dynamically on an otherwise static screen. The design ideology behind it can be expressed as follows: The terminal screen is a small window to the large real world of the game. We can only see a certain offset around the player.

- The code has files like
  - Background.py
  - Headers.py
  - Play.py
  - Utility.py
  - Person.py
  - Board.py
  - Din.py
  - Getch.py
  - Objects.py
- By breaking the code into these files helps us to read the code easily because it means separating it into functions that each only deal with one aspect of the overall functionality.
- Easy to test also like If we were stuck at a point it is easy to see what is happening
- Errors can be handled easily because we have broken the code into many files and we were using mostly calling the functions(modular)
- Reusable code is happening because the codebase seems modular. Duplication of code can be reduced.
- In this codebase we can find the things easily because the same functionality of things is placed in the same file.

Weakness

- It is a bit messy as they splitted things into too many tiny functions and they were passing data or functions around too many files.
- It is not that easy to change the design of the screen and extend the functionalities as few of them were hardcorded.
- The input system's functionality is totally linux based.

System Classes & Responsibilities

System classes	**Responsibilities**
Objects class:	This class is the main(parent) class for all the objects that appear on the board or screen. Parameters: x and y coordinates of an object in the board
Beam class	Inherited from the object class has parameters shape, angle, kill, onboard, canplace along with x and y. It has following functions • Create beam ○ Creating the beam with the given length from shape size • Check beam ○ Checking whether the beam is there or not given x,y,grid and flag value ○ We iterate through the length or size of the beam in the grid(2d array) and return 1 if there is a beam else 0. • Place beam ○ With given angle and grid, we create a beam with the x and y coordinates and place it on the board. While placing , we also check for other beams in that neighbouring to avoid crossing of beams. We place the beam according to the given angle. • Print beam: ○ For printing the beam, first we get x and y coordinates and angle of the beam, and based on that angle we print the beam. Printing means placing the beam on a grid. We have our beam ready, we place it on the grid to print on the board or screen. • Clear beam: ○ Removing the beam from the board . for that based on the angle, we just replace the positions in the grid with a space.

Coins class	This class is for all the coins that appear on the board. It is also inherited from the object class. It has the parameters shape and vis along with x,y coordinates. It has functions get_vis and set_vis for making the coin visible on the board.
Bullet class:	This class is for bullets, It is also inherited from the parent class object. It has the parameters shape,start,active,crash along with the x and y coordinates It has the following functions • set_crash ○ Enabling the bullets to crash, assigning the x value to crash • show_crash ○ Returning bullet's crash value • clear_bullet ○ Clearing the bullet on the boarding. That means replacing it with empty spaces • shoot ○ Shooting the bullets. • Check_collision ○ Checking if the bullets collides with any beams or dragon or player and doing correspondingly.
Powerup class	This class is for powerups, It is also inherited from the parent class object. Has the parameter shape along with x and y coordinates
Magnet class	This class is for Magnet, It is also inherited from the parent class object. Has the parameter shape along with x and y coordinates

Ice Balls class	This class is for Ice Balls, It is also inherited from the parent class object. Has the parameter shape,active status along with x and y coordinates It has the following functions: • Set_active ○ Making the iceball status active by setting value x to iceball active parameter • Get_active ○ This function is to return ice balls active status. • Clear_iceball ○ Clearing the ice ball on the board. Replacing it with empty spaces • Shoot ○ Shooting the iceball • Check_collision ○ Checking if it collides with any bullet or any object in the board and doing correspondingly.
Class Background	This class if for the background for our game It has two parameters ceil and floor Each contains text with style to print It has the following functions • display_floor() ○ To display floor on the screen • display_ceil() ○ To display ceil on the screen
Class Board	Board class inherits from the parent class and it has rows,cols,grid,flag parameters which it uses to create the entire board. It has the following functions • create_board ○ Creates the board (matrix)with given with given rows and columns • Print_board ○ For printing the board on the screen

Class din	Class Din is inherited from the Person class which is imported from the person module. Din has body,body_np,body_shield,body_s,lives,coins,shield_flag,fly_flag,powerflag,mmode,shield_start_time,magnet_flag,last_shield_time,power_start_time,drop_air_time attributes and we have functions to set and show the various flags,power,coins,time,etc. It contains the following major functions ● start_pos() ○ In this function we place the din correctly in the beginning of the grid ● din_clear() ○ In this function, as the din moves it clears the position of the previous move ● din_show() ○ In this function,as we clear the previous position of din we will update din by this function. ● gravity() ○ In this function, if the y pos of din is less than 35 then we check for collision if it collides with any object then we clear the position of din and we will update in new position by decreasing lives if not collided we will update y position by calling sety function. ● Add shield() ○ In this function we will provide a shield to din which protects from enemies
Class Person	This class has two parameters x,y which are coordinates It has the following functions ● getx() ○ For getting x coordinate of the person ● setx() ○ For updating x coordinate of the person ● getty() ○ For getting y coordinate of the person ● sety() ○ For updating y coordinate of the person

Class dragon	This class inherits from the Person class and the parameters are lives,shoot_start and kill. It has the following functions ● dragon_clear() ○ Clears the position of dragon ● dragon_show() ○ This function will update the dragon's new position after it is cleared. ● dragon_move() ○ Dragon moves as y position of it changes and by updating it and showing the pos of dragon

Code Smells

The original code has been beautifully written using a strong notion of OOPs concept. Hence, it was very difficult to pickout code smells from it. But we extracted some of them and they are:-

S.No.	**Code smell**	**Description**

1	Long Method	In **utility.py** file the following function can be shortened to improve modularity in the code.
		def movedin():
		    # moves the player
		Like each input character function can be made and called in this function conditionally.

2	Duplicated Code	Function like gameover() & yay() in file **headers.py** can be shortened using loops like for and contains duplication notions.
3	Dead Code	There is a useless file **`temp.py`** which hasn't been used anywhere else in the code directory.
4	Conditional Complexity	In **utility.py** file function movedin() :Lots of duplicate code in if-else condition block. Rest seems okay in this class
5	Data clumps	**Objects.py :** Every object like Magnets, Ice Balls, Power Ups have their shape hardcoded. One can make a class named “ObjectShape” and then call the required function to access the corresponding object shape.

6	Data class	**Objects.py :** Class PowerUp has stored only the shape of the power up object. This passively stored data does not get operated.
7	Speculative Generality	**All files that contain classes :** Every class that contains shapes like classes for Din, bullets etc possesses the property of coordinate system. We can make a generalized class “Coordinate” which contains x & y values for a shape to get rendered on the Board.
8	Library import	**Background.py, Board.py, Din.py, objects.py, person.py :** “ from headers import * “ : This line will import all the data in ‘headers’ module out of which only few will be used. Therefore one can limit use of `*` and replace with actual constants or functions to be used and only import those.
9	Lazy class / freeloader	**Objects.py :** Class Magnet() inherit class Object() and only have shape attribute effectively. It does too little. And can be removed using a conditional variable.

Code Bugs

S.No.	Category	Description
1	**Major :** File missing	On running the code for the first time got an error: **Alarm exception error.** Later we found out we needed a file to account for that exception handling.
2	**Major :** File missing	On running the code we often get lines printed regarding : coin.wav file missing. Later, we realised that sound functionality is implemented but no files for sounds are available. Basically `./sound folder not found`.

3	**Minor :** Collision handling	Collision of bullets with the boss enemy that Mandalorian fire is not satisfiable. It disappears before colliding with Boss' enemy.
4	**Minor :** Edge Case	When the Boss enemy shoots some item, it consistently removes the coins available on the screen on whichever it gets passed.
5	**Minor :** Inconsistent	Inconsistency in collecting coins- sometimes it gets collected sometimes not by the Mandalorian.
6	**Minor:** Malfunction	Beam of \`#\` character normally decreases the life of Mandalorian. It is very inconsistent in it's working. Also, when the speed of the game fastens that is due to catching the object of below shape, that beam totally loses its functionality. +++ +++
7	**Minor :** Inconsistent	Mandalorian life gets decreased on colliding with the following character that Boss enemy shoots. (@) Sometimes it decreased by 3 units, sometimes 1 unit and sometimes it did not change.
8	**Fatal :** Platform dependency	The frame size i.e. board to display the content has been made very large. Therefore,it cannot get played on Rajat's pc with a smaller screen than expected.
9	**Minor :** Collision handling	When Boss' enemy shoots the bullet, and our mandalorian gets collided with, it should disappear. Rather it passes below it and gives a feeling of mandalorian hovering over it. (@)

10	**Major :** Game Finish	Whenever our Mandalorian gets collided with the left wall, while sticking with an object shown below i.e. magnet, the game finishes which should not happen ideally. `  XXXX` `XXXXXX` `XX   XX` `XX   XX`
11	**Minor :** Collision handling	Whenever our Mandalorian gets collided with Beam of `#`, it's position gets deflected somewhat.

Refactoring

Possible Changes & Benefit

The main goal of refactoring a codebase is to make it more extensible to new features and flexible for future development. There we should try to divide each & every module to the maximum possible extent and remove hardcoded variables like at some point of the code we discover the screen size to be 200 & width 500. Though it is not hard coded but written code does not account for it's variability. We propose to design the components in a way that they scale according to the board size.

We suggest a `Status` class for the game that wraps the whole game functionality. This `Status` class contains information about the player like score, remaining life, time played, level. This will cause implementation of many levels/stages in game leading to incorporated extensibility & extra functionality of code later.

We would propose to alternate the way the code prints the output. Currently each component prints its section separately. This ability that a failure in executing a category technique of an issue could result in a fault display screen output. This can be without difficulty treated by means of retaining a world listing that is updated through each component. This ensures that the display is usually printed. Advantages of the latter are - ease of debugging, quicker refresh rate when you consider that the display screen can be printed after each update, less bulky code and extra manipulation over the output.

Incorporated Design Pattern

- BUILDER DESIGN PATTERN
- DECORATOR DESIGN PATTERN
- SINGLETON DESIGN PATTERN
- OOPS DESIGN PATTERN

The system follows the model view controller design pattern. The _getChUnix class works as a controller in it. As all the components are printed separately they act as a view and the utility which have global functions and variables act as a model which alters the data of the components.

The coin class has four functions that are creating coins. A good way could be making the orientation as an attribute and this could reduce the redundancy of doing the same thing. This is implementing the BUILDER DESIGN PATTERN .

Object class and person class are inherited in the other classes like dragon, beam etc. as shown in uml diagram . This type of design pattern allows us to add new functionality in the child class without altering the parent class and this type of design pattern is called DECORATOR DESIGN PATTERN.

The main code is inside play.py file, basically maingame class is not there, rather the content of play.py should be inside the main.py structured properly inside the class called game. So class games are the composition of players, obstacles, board and dragon. This corresponds to SINGLETON DESIGN PATTERN where a class has only one instance.

There are useless text/quotes present throughout the code that are of no functional importance like while taking an input , the author of the code asks for input and defines a variable valid=1 and then checks if valid==1, then he does the action. This should be removed.
The design pattern is ambiguous as some classes are kept in different files and some are kept in the same file rather than the classification on the basis of their functionality . More the similarity in the code and functioning more should be the code reused to remove redundancy
The whole new restructured code exhibits the OOPS DESIGN PATTERN

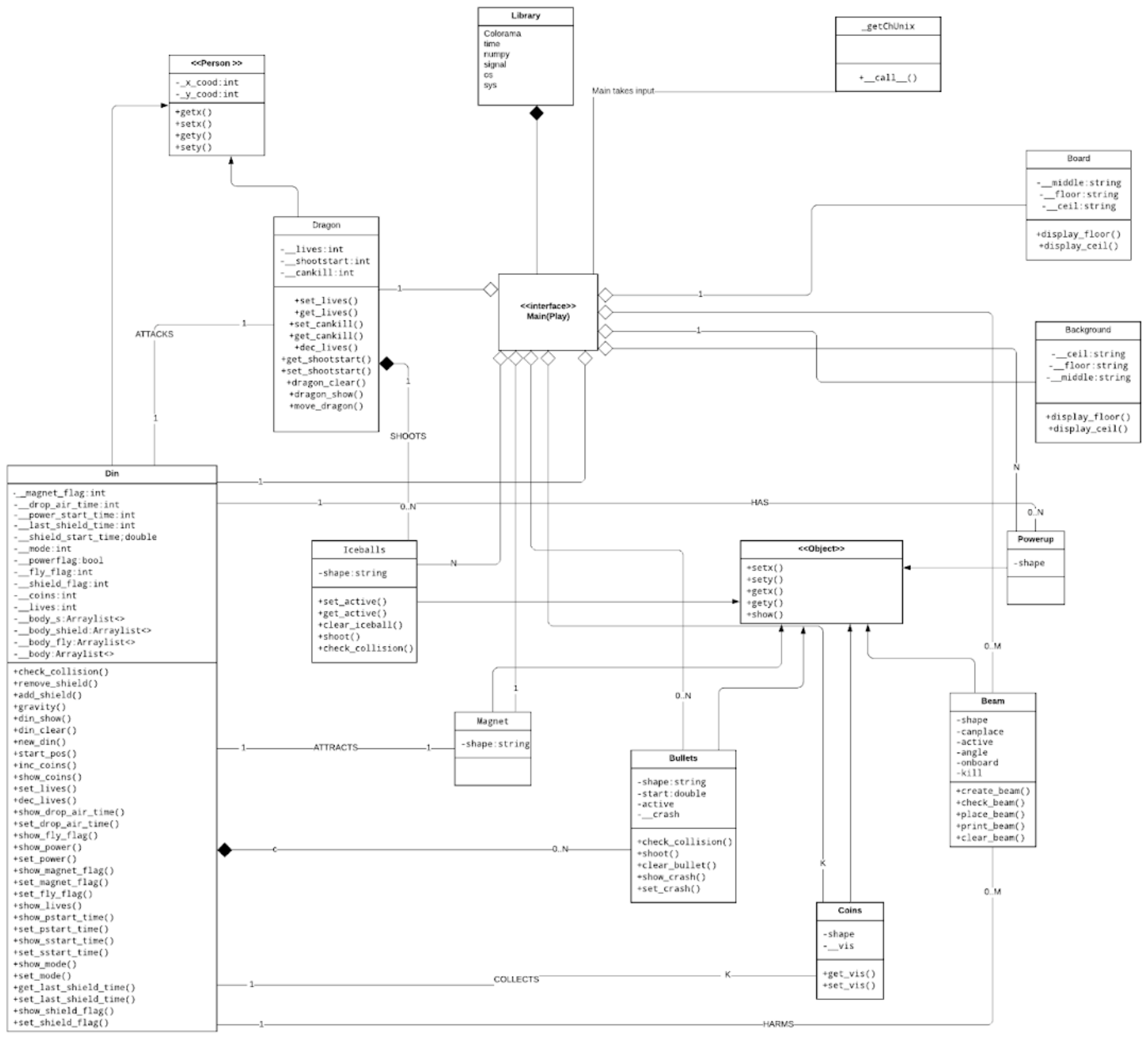
We have noticed a lot of redundancy in the code and two classes with the same functionality works as the same class . Eg: Person and the object class , both are inherited by other classes but we can see that both the class performs the same functions. So both these classes Person and Objects can be merged. The derived classes followed a similar flow for the inherited functions but the was described in each class, rather than being inherited. All the global variables and global functions were intermixed in the utility file.This should be modular and should be created differently.There should be a game class that combines all the functionality of the game . The global function created now should be the functions of the game class. This class would handle all the functionality of the game. This thing helps a lot in making extensible games and when the user enters in a directly another type of scenario when he crosses a particular level. If every level contains the same type of obstacles and functionalities then the user will lose interest in the game. Use of a few libraries was done eg: colorama, numpy etc are mentioned in the uml diagram. The code would be classified into independent components i.e. existence of one will not be defined by another . All these components are stated below.

1) Coins
2) Magnet
3) Beam
4) Board
5) Background
6) Dragon ← Iceball
7) Din ← Bullet

Every other component could be removed or altered without changing another.
Every class has the same type of inherited class but the code could become more modular if class would be divided on the basis of their properties. We could create a parent class named shoot and inherit it in Bullet and Iceball class. It would reduce the code size. If objects of this type were to be required in future like guns then the gun class would directly import shoot class and will automatically have all the functioning of the real world object to shoot in a particular direction.

<u>UML diagram of provided system</u>

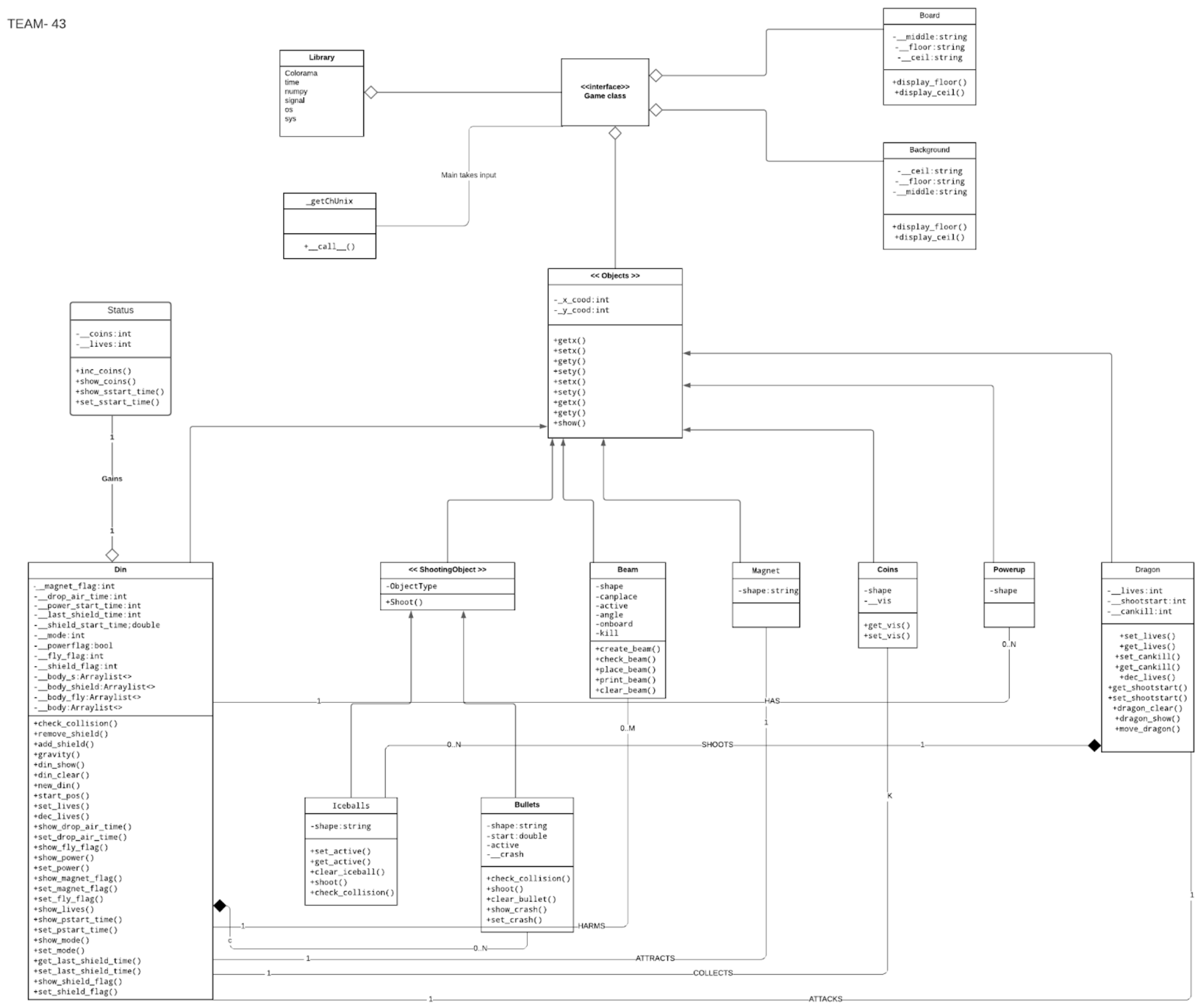
TEAM- 43


Library
Colorama
time
numpy
signal
os
sys
_getChUnix
+__call__()
Main takes input
<<Person >>
-_x_cood:int
-_y_cood:int
+getx()
+setx()
+gety()
+sety()
Dragon
-__lives:int
-__shootstart:int
-__cankill:int
+set_lives()
+get_lives()
+set_cankill()
+get_cankill()
+dec_lives()
+get_shootstart()
+set_shootstart()
+dragon_clear()
+dragon_show()
+move_dragon()
<<interface>>
Main(Play)
Board
-__middle:string
-__floor:string
-__ceil:string
+display_floor()
+display_ceil()
Background
-__ceil:string
-__floor:string
-__middle:string
+display_floor()
+display_ceil()
ATTACKS
SHOOTS
HAS
1
0..N
N
Din
-__magnet_flag:int
-__drop_air_time:int
-__power_start_time:int
-__last_shield_time:int
-__shield_start_time;double
-__mode:int
-__powerflag:bool
-__fly_flag:int
-__shield_flag:int
-__coins:int
-__lives:int
-__body_s:Arraylist<>
-__body_shield:Arraylist<>
-__body_fly:Arraylist<>
-__body:Arraylist<>
+check_collision()
+remove_shield()
+add_shield()
+gravity()
+din_show()
+din_clear()
+new_din()
+start_pos()
+inc_coins()
+show_coins()
+set_lives()
+dec_lives()
+show_drop_air_time()
+set_drop_air_time()
+show_fly_flag()
+show_power()
+set_power()
+show_magnet_flag()
+set_magnet_flag()
+set_fly_flag()
+show_lives()
+show_pstart_time()
+set_pstart_time()
+show_sstart_time()
+set_sstart_time()
+show_mode()
+set_mode()
+get_last_shield_time()
+set_last_shield_time()
+show_shield_flag()
+set_shield_flag()
Iceballs
-shape:string
+set_active()
+get_active()
+clear_iceball()
+shoot()
+check_collision()
<<Object>>
+setx()
+sety()
+getx()
+gety()
+show()
Powerup
-shape
Magnet
-shape:string
ATTRACTS
Bullets
-shape:string
-start:double
-active
-__crash
+check_collision()
+shoot()
+clear_bullet()
+show_crash()
+set_crash()
c
Coins
-shape
-__vis
+get_vis()
+set_vis()
K
COLLECTS
Beam
-shape
-canplace
-active
-angle
-onboard
-kill
+create_beam()
+check_beam()
+place_beam()
+print_beam()
+clear_beam()
0..M
HARMS

System Analysis Improvement:

The Person and the object are merged in this uml diagram with the functions added of both in one. Shoot class is created which contains common functionality in iceball and bullet class .An different score class has been made.
We propose the following structure. This reduces crosslinking with imports and adds a uniform flow, without redundancy. Each function is now only declared and initialised once. Each parent class has an independent functionality, and all classes are merged to work together in the game class.

<u>UML digram of new system design</u>



Individual Contributions

Ayush Sharma

Reverse engineered the code to understand what exclusive modules are doing. I also examined the weblog stated in the pdf and studied unique types of code smells and how they can be improved. I contributed every row in section Code smells and Code Bugs and tried to discover possible answers for the bugs. Contributed to `possible changes & refactoring` part in the Refactor section and Narrative Analysis Of Original Design .Also added a few points in the weakness of the original system.

MC Bhavana

Studied the code and wrote an overall overview of the codebase and it's analysis. Also examined the features of the functions implemented.Contributed to overview of system, narrative analysis and written about some classes and it's responsibilities.

M Pravalika

Studied the codebase and examined all the features implemented in the code. Contributed to the classes and it's responsibilities section, gave detailed explanation of each class and the functions in it. Also added a few points in the strengths of original design.

Rajat Kataria

Studied about the dependencies each file has on others. Studied the code flow, examined system design and found out the flaws in system design .Followed Oops concept and studied the solid principles in order to improve the system design and style in which the classes were made. Tried to remove as much as dependency and crosslinks and suggested a design which will make the system more extensible in future. Checked different design pattern in the system and performed the refactoring of the project. Made uml diagram for initial project and after refactoring.

-----THE END-----