**Game Logic Subsystem**

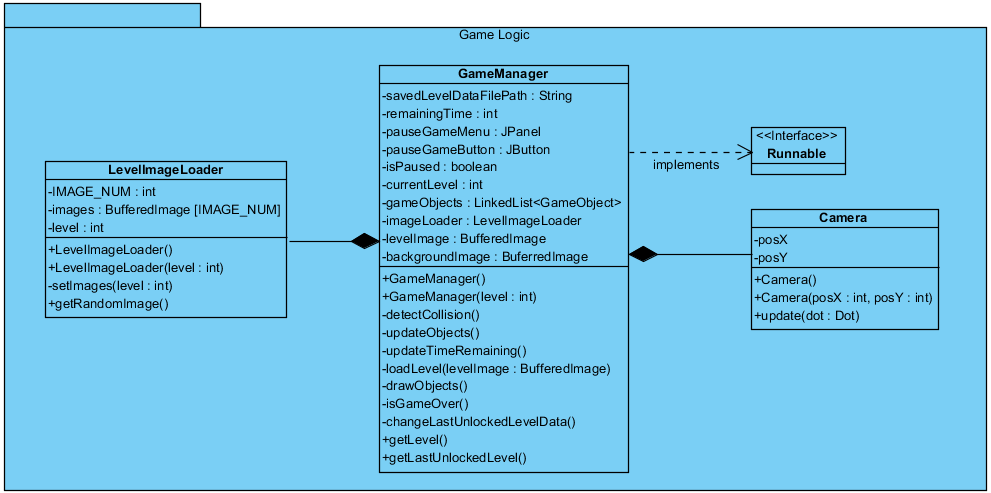
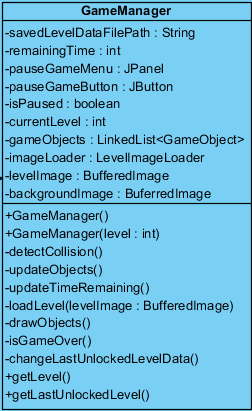


Figure: Package diagram of the Game Logic Subsystem

The figure illustrates the overall composition of the Game Logic Subsystem. The Game Logic Subsystem is responsible for handling and creating the objects of the game. In this subsystem, the positions of the objects will be specified and drawn in the screen.

**Game Manager Class**



The GameManager class is the Façade class of the Game Logic subsystem, thus it is responsible of the creation of the objects that are going to be used in the game. A thread will be running in this class, so it will implement the interface “Runnable” in order to achieve this.

In the GameManager, the attribute savedLevelDataFilePath will be a string, which will contain the file’s path of where the last unlocked level’s data will be written. As the player unlocks levels, the last unlocked level’s number will be written to this file by using the changeLastUnlockedLevelData() method. Also, the public method getLastUnlockedLevel() will be used by the MainMenu class (which was explained in the user interface subsystem) to give the last unlocked level data to the LevelsPanel class (which was also explained in the user interface subsystem).

The GameManager class will be initialized by giving the level’s number to the constructer. By doing this, the GameManager will have the information of the level to be started. The image backgroundImage will be used to specify the locations of the objects such as Dot, letter boxes, spikes etc.

In this class, there will be a linked list of type GameObjects, which will hold all of the GameObjects, for example the Dot, letter boxes, fading letter boxes etc. All GameObjects will be added to this linked list to achieve efficiency.

A short description of the attributes and the methods in the GameManager class to clarify the functionality of each attribute and method:

**Attributes:**

* **savedLevelDataFilePath:** This is a string which specifies the path of the file that will be used to hold the last unlocked level information.
* **remainingTime:** This is an int which holds the remaining time.
* **pauseGameMenu:** This will be the JPanel which will appear when the pause button is clicked.
* **pauseGameButton:** This is the button which will be pressed to pause the game during game play.
* **isPaused:** A Boolean attribute which will keep the data of whether the game is in pause.
* **currentLevel:** An int which will keep the current level number.
* **gameObjects:** This is a linked list of type GameObjects. The objects Dot, LetterBox, FadingLetterBox, Spike and Eraser will all be placed inside this linked list. A specific object will be reached by using a for loop.
* **imageLoader:** This is an LevelImageLoader object which will be used to load the image for a specific level.
* **levelImage:** This will be a BuferredImage which will be the image of the level. This image will be used to specify the positions of the objects in the class. (This is not the image that will be the background picture, it will be used to specify the locations of the GameObjects. Detailed information will be given when explaining the loadLevel method)
* **backgroundImage:** This will be a buferredImage will be the picture to be placed as he background picture.

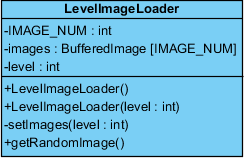
**Constructors:**

* **GameManager:** This is the default constructor.
* **GameManager(level: int):** This is the constructor where the level number information will be given, thus the game will be initialized according to the level.

**Methods:**

* **detectCollision():** This method checks the collision between the Dot and the other GameObjects.
* **updateObjects():** This method will update the positions of the GameObjects in the class.
* **updateTimeRemaining():** The method which will update the remainingTime attribute accordingly.
* **loadLevel(levelImage: BufferedImage):** This method will take the attribute levelImage and specify the locations of the objects. This will be done by using the RGB values in the image. There will be different RGB values representing different GameObject types. This method will go through each pixel and when it comes across a specific RGB value, it will create and place the specific GameObject at this position. For example, whenever it comes across the RGB value (0, 0, 0) which is black, it will create and place a LetterBox object at the specific position.
* **drawObjects():** This method will draw each object in its updated location.
* **isGameOver():** This method will check whether the game is over by checking whether the time is finished and by checking whether the Dot has reached to the end of the sentence.
* **changeLastUnlockedLevelData():** This method writes the last unlocked level’s number into the file which has the path savedLevelDataFilePath. After the game is won this method will be used to update the data in this file.
* **getLevel():** This method will be used to get the current level number.
* **getLastUnlockedLevel():** This method will read the data from the savedLevelDataFilePath attribute and return the last unlocked level. This method will be public so that the MainMenu class could use this method and get the last unlocked level information to pass it on to the LevelsPanel.

**LevelImageLoader Class**



This class is responsible of providing the specific level image to the GameManager class. There will be a specific number of images for each specific level which will be held in an array of BuferredImages. A random level image will be given to the GameManager class. The reason for having different numbers and different types of level images is to present a variety of different game plays even for the same level for the player.

A short description of the attributes and the methods in the LevelImageLoader class to clarify the functionality of each attribute and method:

**Attributes:**

* **IMAGE\_NUM:** This will be a final int value which will hold the number of images that will be held in the images array.
* **images:** This will be an array of size IMAGE\_NUM, which will hold the different level images for the specific level.
* **level:** This will be an int value which holds the number of the level.

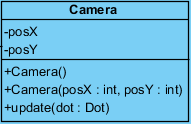
**Constructors:**

* **LevelImageLoader():** This is the default constructor.
* **LevelImageLoader(level: int):** This will initialize the array images according to the given level value. This will be done by using the setImages method.

**Methods:**

* **setImages(level: int):** This method is responsibleof filling the images array when the constructor calls this method. This method will contain several images and according to the level, the images will be set in the images array.
* **getRandomImage():** This is a public method which is used by the GameManager class to get a random image from the images array and set it as the levelImage attribute of the GameManager class.

**Camera Class**



The Camera class is responsible for the movement of the screen which is specified in the GameManager. The camera will have an initial position and the position of the camera will be updated according to the position of the Dot’s position (As the Dot comes to the middle of the screen, the camera will move forward).

A short description of the attributes and the methods in the Camera class to clarify the functionality of each attribute and method:

**Attributes:**

* **posX:** The x coordinate of the camera’s position.
* **posY:** The y coordinate of the camera’s position.

**Constructors:**

* **Camera():** The default constructor.
* **Camera():** The constructor which initializes he camera class with the given x and y coordinates.

**Methods:**

* **Update(dot: Dot):** This method updates the position of the camera. It uses the position of the dot object to update the camera’s position.