

Project: Let’s take a break

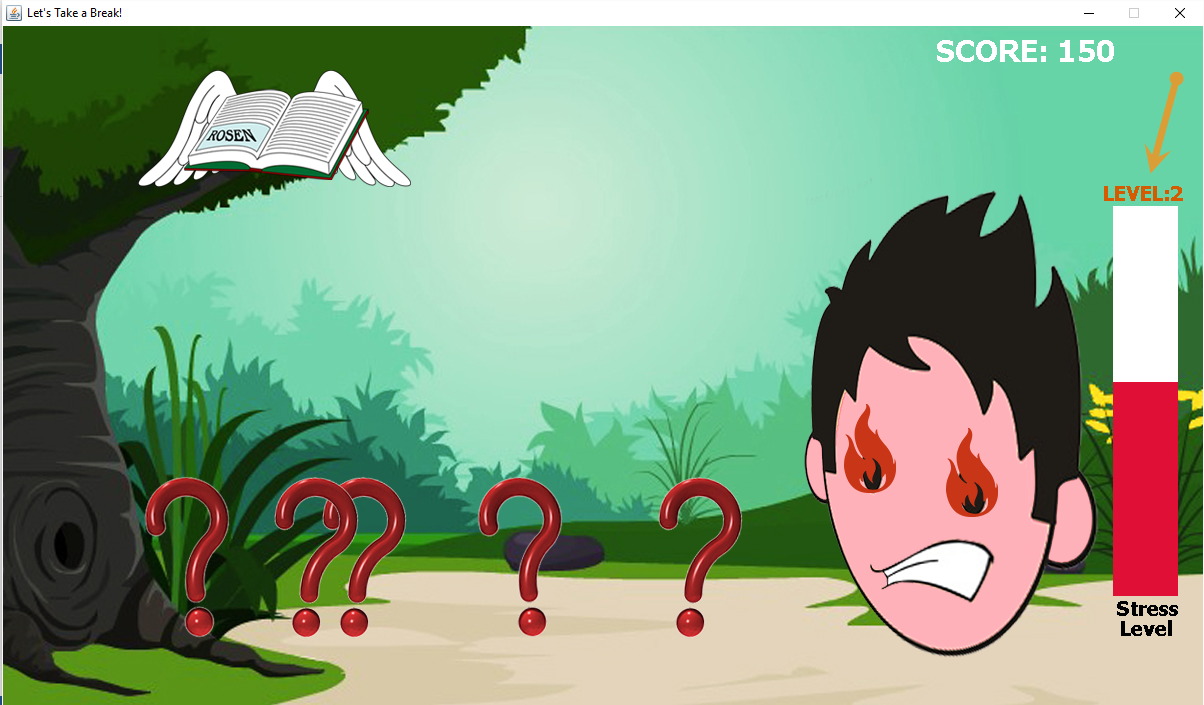


Let’s take a break is a defense game that you have to help one student to eliminate all the stressful things that are very annoying while he is taking a break.

**How to play**

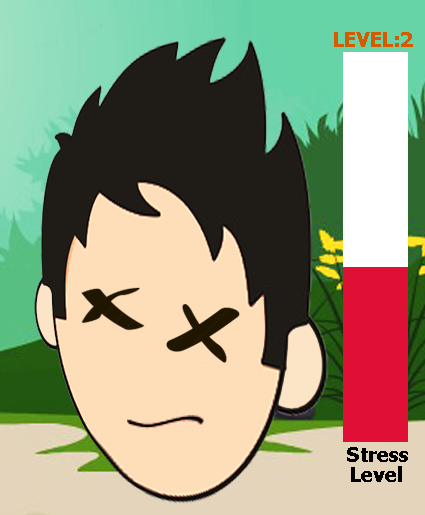
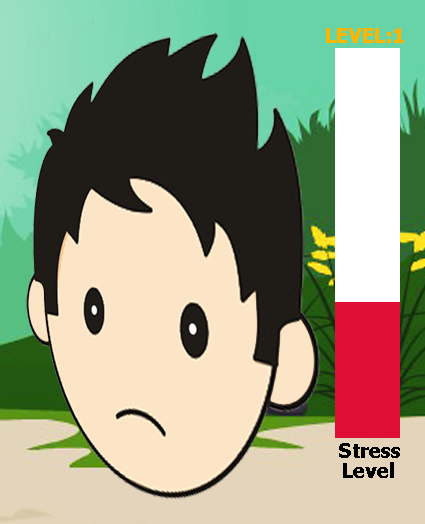
You can help protect him from all the stressful things by clicking all the enemies (figure3) to wipe it but be aware of helpers (figure4), don’t click it. Just let it come to him then they will have some effect.

Every time he was attacked by enemies, it will increase his stress level as shown in Figure 1. When his stress level approaches 100, it will make him, crazier. If his crazy level is over 3, the game is over.



**figure1**

Each of the crazy level can also be observed by seeing student face as shown in figure 2



**Figure2**

These are all the enemies and their effect shown in figure 3.

**Figure3**

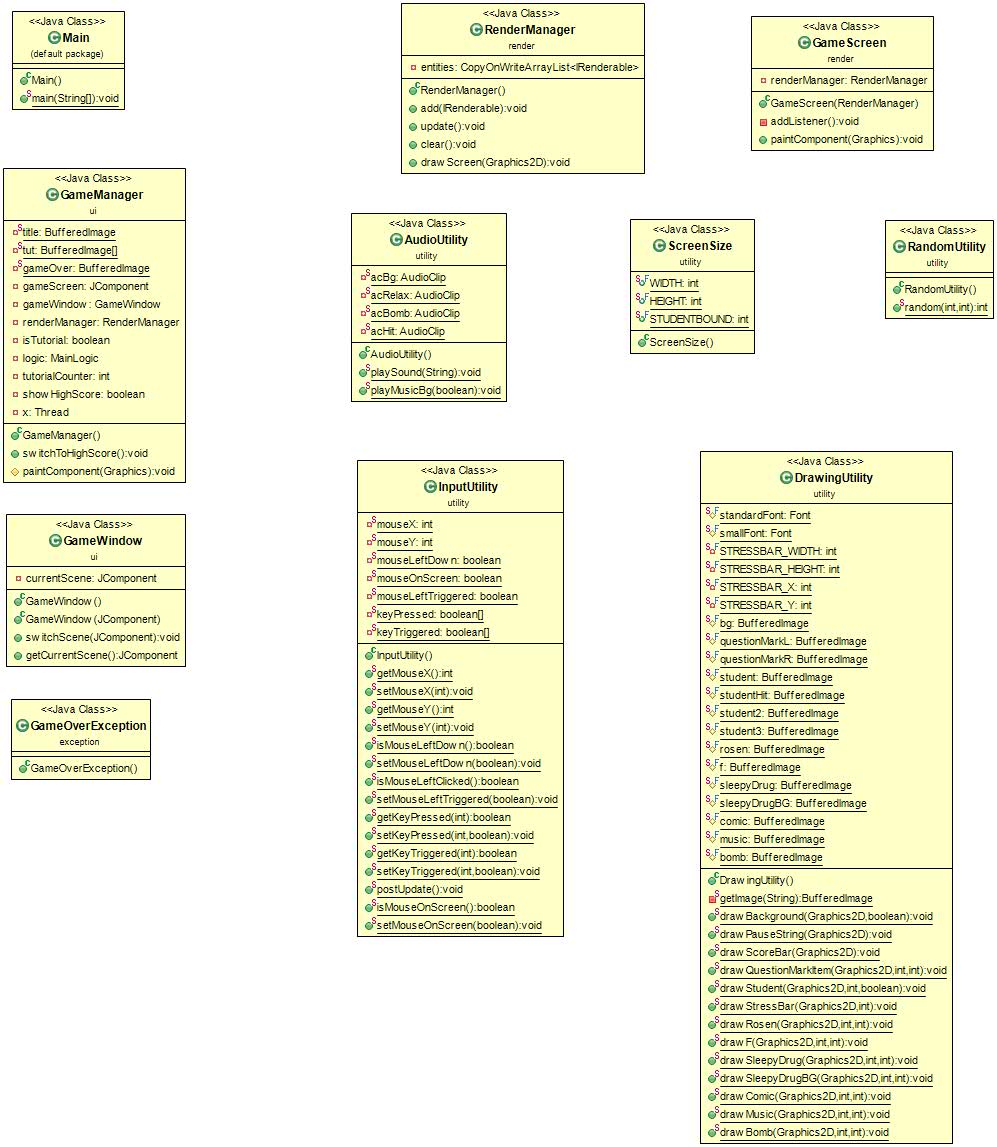


These are all the helpers and their effect shown in figure 4.

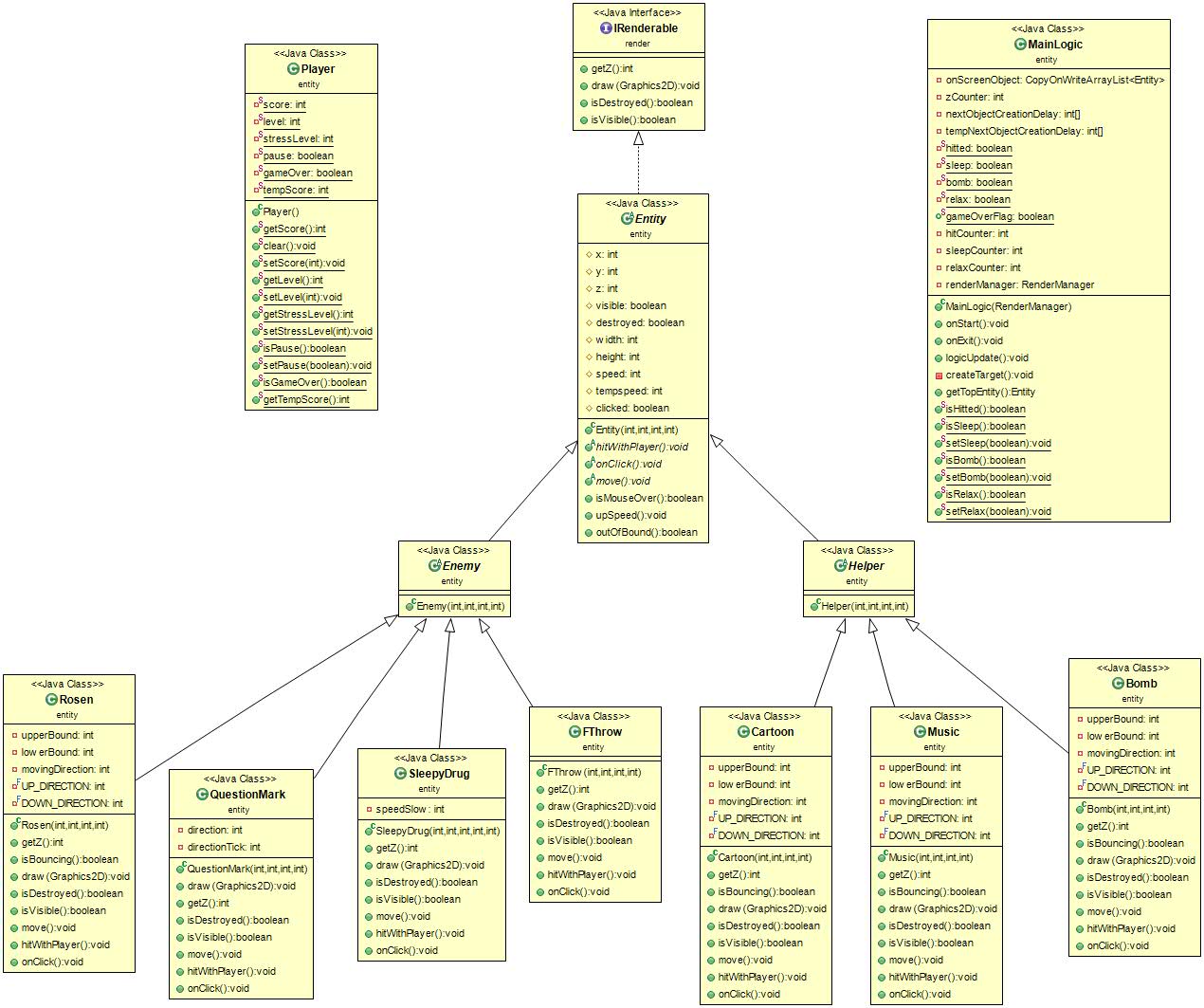
**Figure4**



**UML Diagram**



**Structure of classes in Package “render, utility, ui, exception”**



**Structure of classes in Package “entities”**

**Implementation Details**

1. **Package utility**

There are four classes to help MainLogic and RenderManager easily implemented such as, drawing status bar,play hit sound etc.

1.1 Class DrawingUtility

It is use to draw most past in the game screen by using Graphics2D.

**Field**

* static final Font standardFont ;This is bold Tahoma font with size 30.
* static final Font smallFont ;This is bold Tahoma font with size 20.
* static final int STRESSBAR\_WIDTH = 65;
* static final int STRESSBAR\_HEIGHT = 390;
* static final int STRESSBAR\_X = 1110;
* static final int STRESSBAR\_Y = 180;

This final integer value is described below in the method drawStressBar(Graphics2D g2,int stressLevel)

----------------

* BufferedImage bg ; Load the background;
* BufferedImage questionMarkL ; Load questionMark Entity Left Tick.
* BufferedImage questionMarkR; Load questionMark Entity Right Tick.
* BufferedImage student ; Load student picture level 1.
* BufferedImage student2; Load student picture level 2.
* BufferedImage student3; Load student picture level 3.
* BufferedImage f; Load the student hit picture.
* BufferedImage sleepyDrug; Load the sleepyDrug entity picture.
* BufferedImage sleepyDrugBg; Load the sleeping background.
* BufferedImage rosen; Load the rosen entity picture.
* BufferedImage comic; Load the comic entity picture.
* BufferedImage music; Load the music entity picture.
* BufferedImage bomb; Load the bomb entity picture.

All the Field above is loaded using Class Loader By Method getImage(String directory);

**Method**

* static BufferedImage getImage(String directory); This method uses class loader to get image and return in BufferedImage.
* static void drawBackground(Graphics2D g2,boolean sleep); This methods will draw background(bg) if sleep is false. else it will draw sleepyDrugbg.
* static void drawPauseString(Graphics2D g2); This method will draw “PAUSE” string in the center of the Screen if Player.isPause is true.
* static void drawScoreBar(Graphics2D g2); This method will draw “SCORE: ” +Player.getScore() at the (ScreenSize.WIDTH\*7/9,0).
* static void drawQuestionMarkItem(Graphics2D g2,int x,int direction); If the direction is 1 it will draw questionMarkL at (x,450). else it will draw questionMarkR at the same coordinate.
* drawStudent(Graphics2D g2,int level,boolean playerIsHit); if player is hitted it will draw studentHit. Otherwise, it will draw student1-3 by level of student.
* static void drawStressBar(Graphics2D g2,int stressLevel); This method will draw red stress bar at the top left with the red filled stressLevel (max 100) at coordinate(STRESSBAR\_X,STRESSBAR\_Y). The width and height of full bar is STRESSBAR\_WIDTH and STRESSBAR\_HEIGHT. It also draw the string using standard font with black color at (STRESSBAR\_X+4,ScreenSize.HEIGHT-90).

--------------------------------------------------------------------------------

* static void drawRosen(Graphics2D g2,int x, int y);
* static void drawF(Graphics2D g2,int x, int y);
* static void drawSleepyDrug(Graphics2D g2,int x, int y);
* static void drawSleepyDrugBg(Graphics2D g2,int x, int y);
* static void drawComic(Graphics2D g2,int x, int y)
* static void drawMusic(Graphics2D g2,int x, int y)
* static void drawBomb(Graphics2D g2,int x, int y)

All of above methods use to draw image of entity at coordinate (x,y).

1.2 Class AudioUtility

This class is dealing with sound playing in this games.

**Field**

* static AudioClip acBg; Background Sound.
* static AudioClip acRelax; Relaxing Sound.
* static AudioClip acBomb; Bomb Sound.
* static AudioClip acPause; Pause Sound.
* static AudioClip acHit; Hit Sound.

This is audio clip loaded by Class Loader using static way.

**Method**

* static void playMusicBg(boolean isRelax);
* static void playSound(String identifier);

This method will playsound using identifier.

* + Identifier = bg acBg.loop(); will be executed.
  + Identifier = relax acRelax.play (); will be executed.
  + Identifier = bomb acBomb.play(); will be executed.
  + Identifier = pause acPause.play(); will be executed.
  + Identifier = hit acHit.play(); will be executed.

If isRelax is true. acBg loop will stop and acRelax will play in a loop.

else acRelax will stop and acBg will paly in a loop.

1.3 Class InputUtility

This class has an array of boolean to save every key state(if that button was pressed. that boolean will be true. or key released that boolean will be false.

**Field**

* static int mouseX,mouseY; using to save coordinate x,y of mouse.
* static boolean mouseLeftDown; is true when left mouse if pressed.
* static boolean mouseLeftTriggered; is true at first tick when left mouse is clicked.
* static boolean[] keyPressed = new boolean[256]; using to save keyPressed boolean.
* static boolean[] keyTriggered = new boolean[256]; using to save keyTriggered boolean.

**Method**

* There are all getter and setters in all field.
* static void postUpdate(); using to update all keyTriggered to be false.

1.4 Class RandomUtility

This class has static method random(int min,int max) to random integer ranging from min to max.

**Method**

* static int random(int min,int max); This will return random integer ranging from min to max.

1.5 Class ScreenSize

This class stores screen size and students bound(where method drawStudent in class DrawingUtility should draw.

**Field**

* static final int WIDTH = 1200;
* static final int HEIGHT = 680;
* static final int STUDENTBOUND = 4\*ScreenSize.WIDTH/6;

**2.Package ui**

This package is dealing with the gameMenu Tutorial and the gameScreen.

2.1 Class GameWindow

This class extends JFrame to be the main screen.

**Field**

* JComponent currentScene; this is JComponent to be putted in JFrame.

**Constructor**

GameWindow(JComponent scene); This will set this frame title as “Let’s take a break! and will set setDoubleBuffered and setVisible be true. Set currentScene = scene; and add currentScene to gameWindow (Note. use pack and requestFocus to make it correctly drawn)

**Method**

* void switchScene(JComponent scene); remove the old scene in gameWindow and add scene. set currentScene= scene; (Note. using validate requestFocus and repaint to make it correctly work.)

2.2 Class GameManager

This class is extends from JComponent showing menu and to control the flow of the game.

**Field**

* static BufferedImage title ; It will be loaded title image.
* static BufferedImage tut[]; It will be loaded tutorial01 to 04 image.
* static BufferedImage gameOver; It will be loaded gameOver image.
* GameWindow gameWindow; It is the main gameWindow(JFrame that is showing up when run the game)
* RenderManager renderManager; It is renderManager dealing with drawing gameScreen.
* boolean isTutorial; It is Boolean to check which image should be drawn in to the screen.
* MainLogic logic; It is the MainLogic of the game.
* int tutorialCounter; It is counter to count which image from tut[x] should be drawn.
* boolean showHighScore; It will return true if game is over.
* Thread x; It is the thread that will run mainLogic.logicUpdate() loop after new game.

**Constructor**

GameManager() ; This constructor will initialize all field. and creating Thread x will be by creating Annonymous Runnable Class the run() method insides will synchonized with logic It also try logic.wait() until user click play then logic will be notified and do the game loop by Thread.sleep(20) and then repaint the gameScreen and update logic This thread will be teminated after game is over. It will call logic.onExit() and switchToHighScore() screen. This constructor also addMouseListener by creating Annonymous Inner Class.

The MousePressed in inner class will be implements by checking if it is tutorial screen(isTutorial) or (ShowHighScore) It will listen at (1055<=x<=1163,557<=y<=656) if it is clicked at this area if it is tutorial Screen it will count the tutorialCounter up until tutorialCounter exceed 3 isTutorial will false. or if it is high score menu after clicked it will be false. after any changes, repaint(); is need to update the screen.

If it is not tutorial or high score screen it will listen at the coordinate x,y of 3 buttons as shown in menu screen. if play button is clicked . it will notify the logic and gameWindow will switch scene to gameScreen. if help is clicked it will set isTutorial true and repaint() the screen. if exit is clicked. It will close the game.

**Important method**

* void switchToHighScore() ; This method will be called by Thread x after game is over. and gameWindow will switch scene to this class. Thread x will be initialized again by synchronizing with logic and wait for logic to notify and continues game loop until game is over.
* void paintComponent(Graphics g) ; This method will draw Tutorial sceen from tut[tutorialCounter] if isTutorial is true. or this method will draw gameOver image and draw String Player.getTempScore() ; at the center of the screen. if showHighScore is true Otherwise, it will draw the title screen.

**3.Package render**

This package is responsible for rendering every object in the game.

3.1 Interface IRenderable

This is an interface specifying method required for the object that can be drawn onto the screen.

**Important method**

* bool isVisible(); This method returns “true” if the object is visible.
* bool isDestroyed(); This method returns “true” if the object is destroyed.
* intgetZ(); This method returns index of the object in z-axis. Note that object can overlap other objects on the screen. The top one has the highest value.
* void draw(Graphics2D g2d); This method is used for drawing object.

3.2 Class GameScreen

This class is extended from JComponent. It is the game screen of this game.

**Field**

RenderManager renderManager; This is class dealing what to draw in the game screen.

**Constructor**

GameScreen(RenderManager renderManager) ; This will set the properties of GameScreen as listing below

* + setDoubleBuffered(true);
  + setPreferredSize(new Dimension(1200,680));
  + setvisible(true);
  + addListener(); will describe below.
  + initialize renderManager;

**Important method**

* void addListener(); This method will add MouseListener,MouseMotionListener and KeyListener implementing using anonymous inner class and set variables in InputUtility if key ,mouse are pressed or relased.
* void paintComponent(Graphics g); This method will call drawScreen from renderManager to draw the game screen of that state.

3.3 Class RenderManager

This class uses to draw the game screen by updating the entities that needs to be drawn.

**Field**

* private CopyOnWriteArrayList<IRenderable>(); This is all of the entities on the screen.

**Constructor**

RenderManager(); It will initialize the entities;

**Important method**

* void add(IRenderable entity) ; This method will add entity and sort list by Z value( getZ() ).
* void update(); if entities is destroyed. it will remove from entities list.
* void drawScreen(Graphics2d g2d); It will drawn as shown below.

DrawingUtility.drawBackground(g2d,MainLogic.isSleep());

DrawingUtility.drawScoreBar(g2d);

DrawingUtility.drawStudent(g2d,Player.getLevel(),MainLogic.isHitted());

DrawingUtility.drawStressBar(g2d, Player.getStressLevel());

calling entities.draw(g2d); of every visible and not destroy entities;

remove the destroyed entities. (update() )

**4. Package exception**

This package is responsible for all the exception handling in the game.

4.1 Class GameOverException

This is a class, which represents the exception that will work when player’s crazy level is more than 3. This exception will be thrown when the game is over.

**5. Package entities**

This package is responsible for all the entities in the game that are divided in 2 groups which are enemy and helper and also including MainLogic and Player class.

5.1 Class Entity

This is an abstract class, which represents all of the objects in the field.

**Constructor**

GameOverException(); The constructor that will print “The game is over” and also print the score of the player when the game is over.

**Field**

* int x, y, z; The current position (x, y, z) and the direction of the tank
* bool destroyed; An entity will be removed from the field when this variable is true.
* Int width, height; The width and height of the picture of each entities.
* Int speed; the distance when each entities move per tick
* Int tempspeed; The initial speed.
* bool clicked; This variable is true when an entity is clicked.

**Constructor**

Entity(int x, int y, int z, int speed); The constructor assigns values to all field and set visible to be true and destroyed to be false. Also, setting tempspeed equals speed.

**Important method**

* void hitWithPlayer(); This method will make change when entities hit player.
* void onClick(); This method works when each entity is clicked. It will change clicked to be true and set speed and temp speed to be -20.
* void move(); This method will work when entity move.
* void upSpeed(); This method will increase speed when player’s crazy level is increased.
* boolean outOfBound(); this method will return true when it’s move across player’s picture.
* boolean isMouseOver(); this method will return true when mouse is over.

5.2 Class Enemy

This is an abstract class, which represents all of the enemies in the field.

**Constructor**

Enemy(int x, int y, int z, int speed); The constructor assigns values to all super variable.

5.3 Class Helper

This is an abstract class, which represents all of the helpers in the field.

**Constructor**

Helper(int x, int y, int z, int speed); The constructor assigns values to all super variable.

5.4 Class QuestionMark

It represents a question mark, which is an enemy.

**Field**

* Int direction; The animation number of the question mark.
* Int directionTick; The time per each picture of the animation.

**Constructor**

QuestionMark(int x, int y, int z, int speed); It initializes all class/superclass fields.  
- set direction = 1, directionTick = 0, and set width and height equals width and height of the picture.

**Important method**

* void draw(Graphics2D g2d); This method will draw the picture of this entity by calling DrawingUtility.
* void move(); This method will control moving of this entity. Every time this method is called, it will increase 1 directionTick and if directionTick is more than 7, it will multiply direction with -1. Also, increase x with its speed. If x is less than 0, set destroyed to be true. Moreover, calling hitWithPlayer method when it’s out of bound.
* void void hitWithPlayer();This method will make change when this entity hit player by increasing stress level with 10.
* void onClick(); This method works when each entity is clicked. It will change clicked to be true and set speed and temp speed to be -20. Also, increasing player’s score by 150.

5.5 Class Rosen

It represents a study book, which is an enemy.

**Field**

* int upperBound, lowerBound; The bound of the movement of Rosen
* int movingDirection; The direction of Rosen.
* int UP\_DIRECTION, DOWN\_DIRECTION; They are constant values representing direction

**Constructor**

Rosen(int x, int y, int z, int speed); It initializes all class/superclass fields.  
- setting movingDirection to be up direction, upperBound equals its y – 100, lowerBound to be 0 and set width and height equals width and height of the picture. If its upperBound is less than 0, set it to be 0.

**Important method**

* void draw(Graphics2D g2d); This method will draw the picture of this entity by calling DrawingUtility.
* boolean isBouncing(); return true when its y more than lowerBound or less than upperBound
* void move(); This method will control moving of this entity. If it’s bouncing, it will multiply movingDirection with -1. Also, increase x, y everytime it’s called. Moreover, calling hitWithPlayer method when it’s out of bound.
* void void hitWithPlayer();This method will make change when this entity hit player by increasing stress level with 35.
* void onClick(); This method works when each entity is clicked. It will change clicked to be true and set speed and temp speed to be -20. Also, increasing player’s score by 250.

5.6 Class FThrow

It represents a F, which is an enemy.

**Constructor**

FThrow(int x, int y, int z, int speed); It initializes all class/superclass fields.  
- setting width and height equals width and height of the picture.

**Important method**

* void draw(Graphics2D g2d); This method will draw the picture of this entity by calling DrawingUtility.
* void move(); This method will control moving of this entity, increasing x every time it’s called. If x is less than 0, set destroyed to be true. Moreover, calling hitWithPlayer method when it’s out of bound.
* void void hitWithPlayer();This method will make change when this entity hit player by setting stress level to be 80.
* void onClick(); This method works when each entity is clicked. It will change clicked to be true and set speed and temp speed to be -20. Also, increasing player’s score by 100.

5.7 Class SleepyDrug

It represents a sleepy drug, which is an enemy.

**Field**

* int speedSlow; The speed when it’s hit the player.

**Constructor**

SleepyDrug(int x, int y, int z, int speed, int speedSlow); It initializes all class/superclass fields.  
- setting width and height equals width and height of the picture.

**Important method**

* void draw(Graphics2D g2d); This method will draw the picture of this entity by calling DrawingUtility.
* void move(); This method will control moving of this entity. It will increase x by its speed when it isn’t clicked and x is less or equal than 400. If x is more than 400, its x will increase by speedSlow. If x is less than 0, set destroyed to be true. Moreover, calling hitWithPlayer method when it’s out of bound.
* void void hitWithPlayer();This method will make change when this entity hit player by calling MainLogic.setSleep and set destroyed to be true.
* void onClick(); This method works when each entity is clicked. It will change clicked to be true and set speed and temp speed to be -20. Also, increasing player’s score by 350.

5.8 Class Cartoon

It represents a cartoon, which is a helper.

**Field**

* int upperBound; The upper bound of the movement of this entity which equals 0.
* int lowerBound; The lower bound of the movement of this entity which equals height of the screen.
* int movingDirection; The direction of this entity.
* int UP\_DIRECTION, DOWN\_DIRECTION; They are constant values representing direction

**Constructor**

Cartoon(int x, int y, int z, int speed); It initializes all class/superclass fields.  
- setting movingDirection to be up direction and width and height equals width and height of the picture.

**Important method**

* void draw(Graphics2D g2d); This method will draw the picture of this entity by calling DrawingUtility.
* boolean isBouncing(); return true when its y more than lowerBound or less than upperBound
* void move(); This method will control moving of this entity. If it’s bouncing, it will multiply movingDirection with -1. Also, increase x, y every time it’s called. Moreover, calling hitWithPlayer method when it’s out of bound.
* void void hitWithPlayer();This method will make change when this entity hit player by decreasing stress level by 40.
* void onClick(); This method works when each entity is clicked. It will change clicked to be true and set speed and temp speed to be -20. Also, decreasing player’s score by 300.

5.9 Class Music

It represents a music, which is a helper.

**Field**

* int upperBound; The upper bound of the movement of this entity which equals 0.
* int lowerBound; The lower bound of the movement of this entity which equals height of the screen.
* int movingDirection; The direction of this entity.
* int UP\_DIRECTION, DOWN\_DIRECTION; They are constant values representing direction

**Constructor**

Music(int x, int y, int z, int speed); It initializes all class/superclass fields.  
- setting movingDirection to be up direction and width and height equals width and height of the picture.

**Important method**

* void draw(Graphics2D g2d); This method will draw the picture of this entity by calling DrawingUtility.
* boolean isBouncing(); return true when its y more than lowerBound or less than upperBound
* void move(); This method will control moving of this entity. If it’s bouncing, it will multiply movingDirection with -1. Also, increase x, y every time it’s called. Moreover, calling hitWithPlayer method when it’s out of bound.
* void void hitWithPlayer();This method will make change when this entity hit player by decreasing speed of all the entities in the map & calling Bgmusic for 1400 ticks & play relaxing sound. All of these can call MainLogic.setRelax to help work.
* void onClick(); This method works when each entity is clicked. It will change clicked to be true and set speed and temp speed to be -20. Also, decreasing player’s score by 200.

5.10 Class Bomb

It represents a music, which is a helper.

**Field**

* int upperBound; The upper bound of the movement of this entity which equals 0.
* int lowerBound; The lower bound of the movement of this entity which equals height of the screen.
* int movingDirection; The direction of this entity.
* int UP\_DIRECTION, DOWN\_DIRECTION; They are constant values representing direction

**Constructor**

Music(int x, int y, int z, int speed); It initializes all class/superclass fields.  
- setting movingDirection to be up direction and width and height equals width and height of the picture.

**Important method**

* void draw(Graphics2D g2d); This method will draw the picture of this entity by calling DrawingUtility.
* boolean isBouncing(); return true when its y more than lowerBound or less than upperBound
* void move(); This method will control moving of this entity. If it’s bouncing, it will multiply movingDirection with -1. Also, increase x, y every time it’s called. Moreover, calling hitWithPlayer method when it’s out of bound.
* void void hitWithPlayer();This method will make change when this entity hit player by removing all the entities in the map. It will call MainLogic.setBomb to help work and also play bomb sound and set destoyed to be true.
* void onClick(); This method works when each entity is clicked. It will change clicked to be true and set speed and temp speed to be -20, movingDirection to be 0. Also, decreasing player’s score by 500.

5.11 Class Player

It contains all the player status in the game.

**Field**

* static int score; The current score of the player.
* static int level; The current crazy level of the player.
* static int stressLevel; The current stress level of the player.
* Static bool pause; When the game is pause, the variable is true.
* Static bool gameOver; When the game is over, the variable is true.
* Static int tempScore; The score to be shown in high score screen.

**Constructor**

Player(); It initializes all class fields.  
- setting score and stressLevel to be 0.

**Important method**

* +getter & setter
* static void setStressLevel(); This method will set value of the stress level.  
  If stress level is more than 0, it should increase player’s crazy level by 1. If there is GameOverException, set stress level to be 100 and gameOver to be true.
* static void clear(); This methods will be called before switching scene.  
  It will set tempScore equals score and reset every variables to be default value.

5.12Class MainLogic

This class is dealing with every entities movement

**Field**

* int zCounter = Integer.MIN\_VALUE + 1; It is z counter that will count up after add an entity.
* int[] nextObjectCreationDelay ; This initialize all object creation delay.
* static boolean hitted; return true if player is hitted by entities.
* static boolean sleep; return true if player is received sleepyDrug.
* static boolean bomb; return true if player is received Bomb.
* static boolean relax; return true if player is received Music.
* static boolean gameOverFlag; return true if player is reached level 3 with stressLevel 100.

-------------------------------------------------------------------------------------------------

* int hitCounter = 0;
* int sleepCounter = 0;
* int relaxCounter = 0;

Using as a tick counter how long which item should take effect.

private RenderManager renderManager;

**Constructor**

MainLogic(RenderManager renderManager); This will initialize renderManager and call onStart() method.

**Important method**

There are all getters and setters for all field.

* void onStart() ; This method will initialize hitted and sleep be false and play “bg” song.
* void onExit() ; This method will set all field be zero or false and clear all lists and also playMusicBg(false), also set all the nextObjectCreationDelay to the default value;
* void logicUpdate() ; This method can be separated into 3 parts.

I.**Tick counter** to count how many tick they should stay before change the values.

If sleep is true, the sleepCounter will count up until equals 150 then, it will set sleep false and sleepCounter =0.

If relax is true it will count up untils equals 1400 then , it will set relax false and set relaxCounter =0 and playMuusicBg(false);

II.**Update Target** it will call createTarget and update all entites (by calling move method of all undestroyed entities and removing all destroyed entities) if InputUtility.isMouseLeftDown() is true, it will call getTopEntity.onClick();

III.**Pause** If Player.isPause() is true it will return; and if Player.isPause() is false and InputUtility. getKeyTriggered(KeyEvent.VK\_ENTER)) is true it will toggle the pause value.

* Entity getTopEntity() ; It will find onScreenObject that is mouse over and is top entity then, return Entity.
* void createTarget() ; It will decrease all nextCreationDelay[] by one if nextCreationDelay[x] equals zero it will create entities x and add to the onScreenObject and renderManger then, it will random nextCreationDelay and increases zCounter by one.