Frogger

Korranat Naruenatthanaset

5630008021

Navee Sratthatad

5630332221

Project (1/2014) in 2110215 Prog Meth

FROGGER

Frogger is a classic game from the golden age of video games. Frogger is still popular and can be found on many websites. Player needs to help the frog one by one to across the road avoiding the cars and cross the river by jumping on logs and turtles

How to play

Main menu of the game has 2 buttons, play button and best records button.

Best record screen has 3 pages, the scores for easy, normal, and hard level. Each page has top 5 records in each level.


Little Frogger
Best Record
Normal
1 Eric 3 " 7 " 62
2 Sophia 10 " 5 " 98
3 Eve 10 " 15 " 13
4 Edward 15 " 17 " 34
5 May 22 " 13 " 90
BACK


Little Frogger
Best Record
Hard
1 Yegar 10 " 19 " 98
2 Christ 12 " 15 " 13
3 Paul 15 " 13 " 90
4 Mark 18 " 12 " 62
5 Henry 23 " 11 " 34
BACK

Select level screen has 3 button easy, normal, and hard level. Select any button to play the game

In the game, you have 5 lives. Use arrow key (up, down, left, and right) to move the frog, avoiding the cars and cross the river by jumping on logs and turtles then move to gap between trees. To win the game you have to cross the river 5 times.

Press “ENTER” to pause the game.

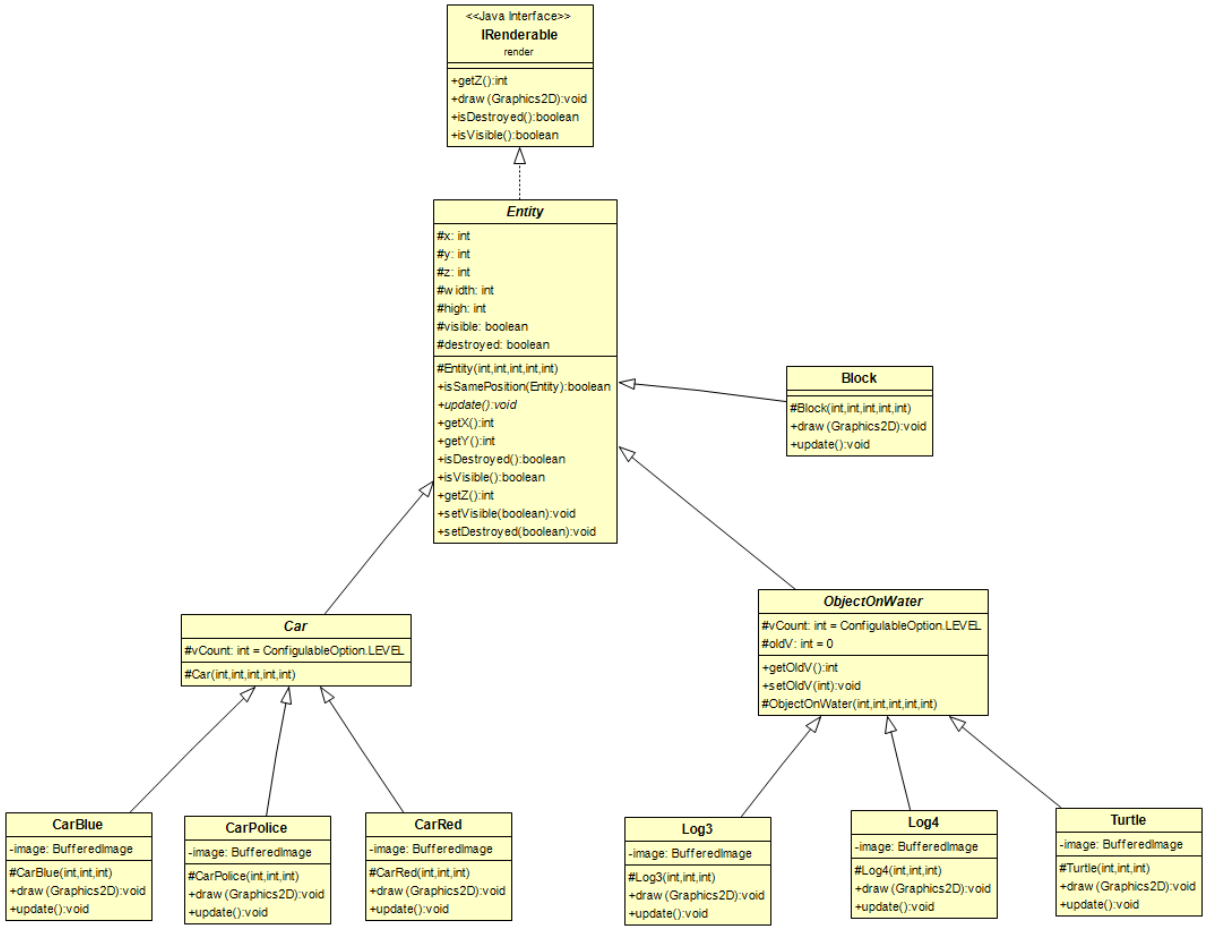
If you lose the game, it will turn to game over screen. To go to main menu just left click 1 time.

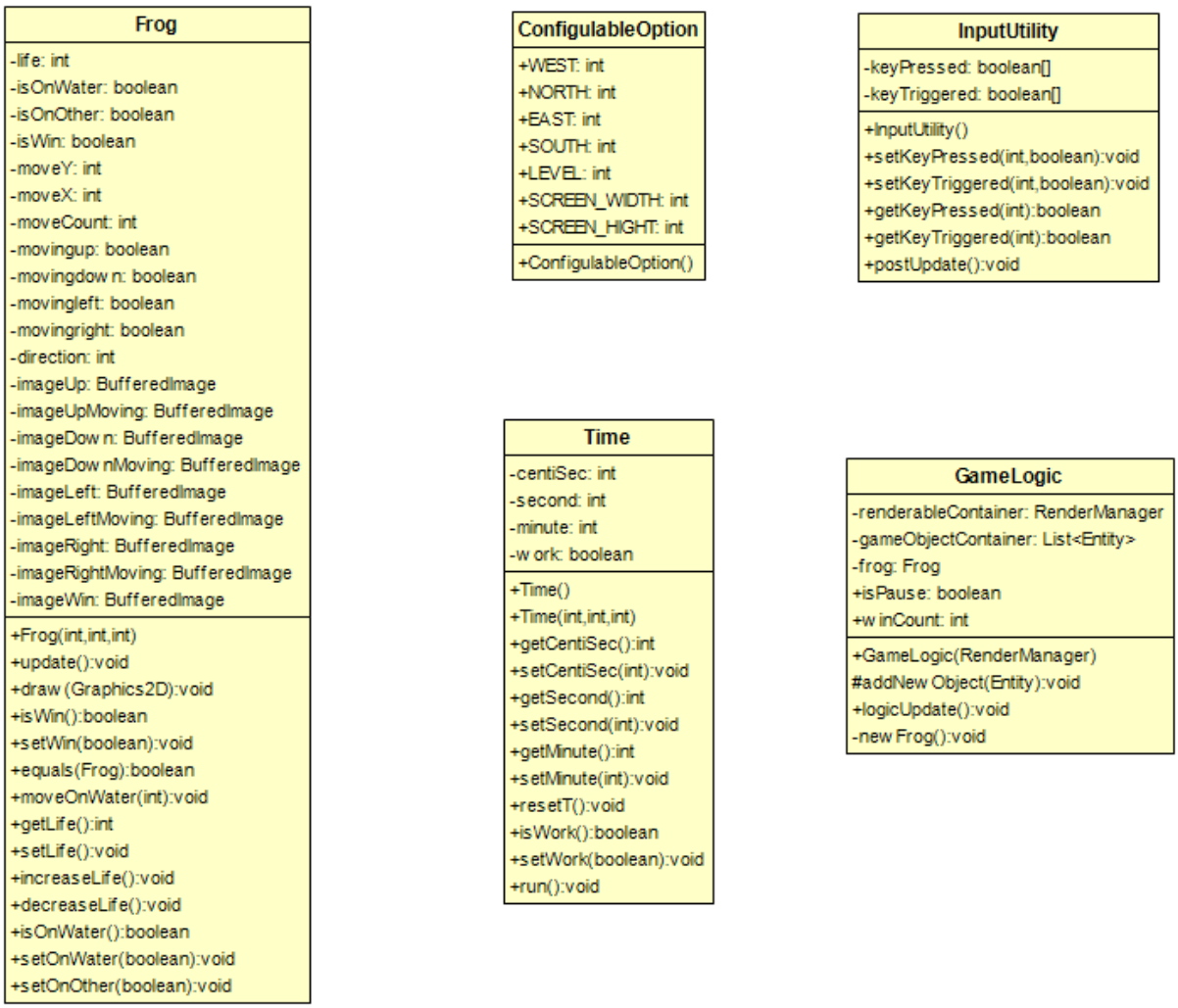
If you win the game, it will turn to win screen and has dialog box to fill your name. To go to main menu left click 1 time.

Implementation Details

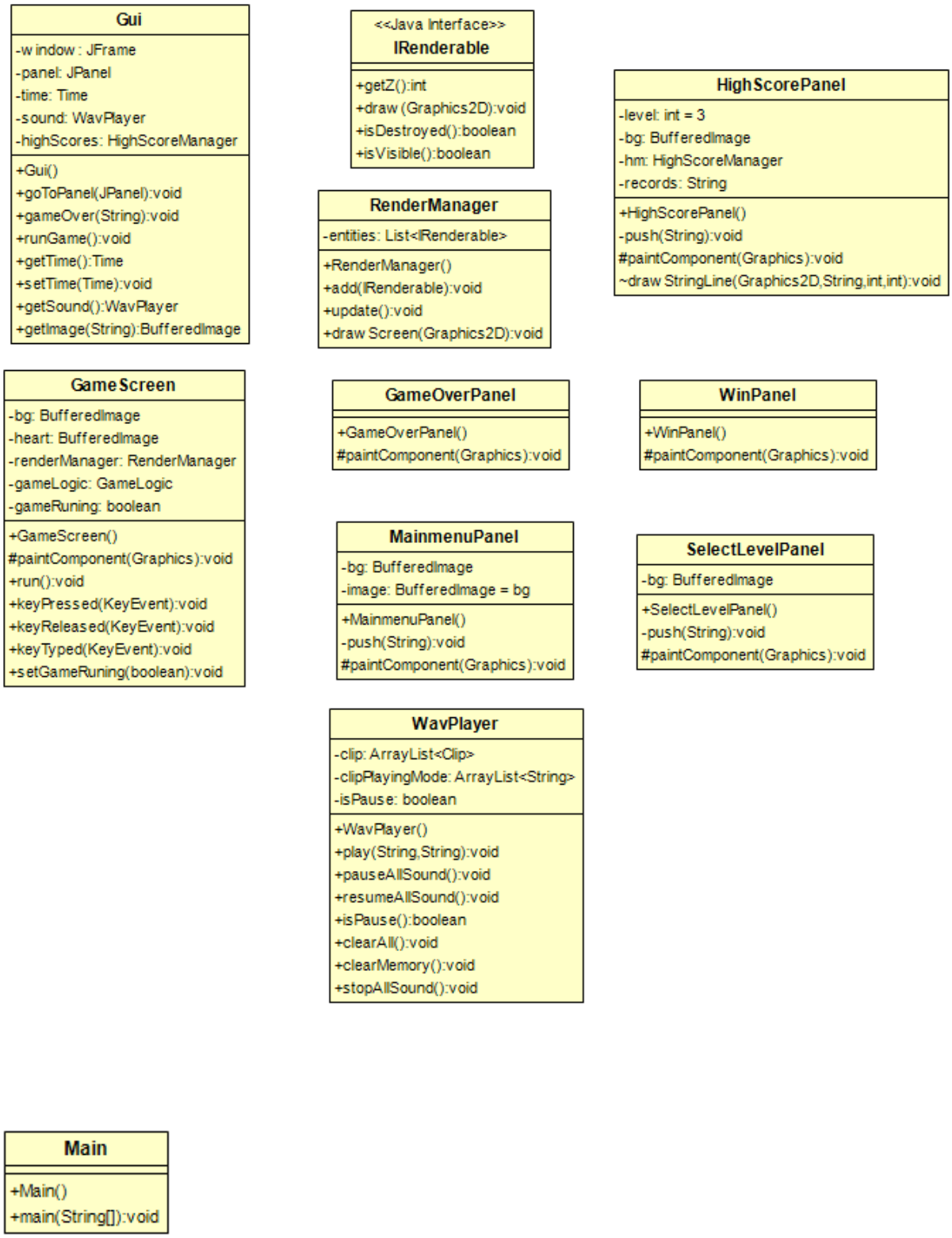
UML Diagram

Package logic




Frog
-life: int
-isOnWater: boolean
-isOnOther: boolean
-isWin: boolean
-moveY: int
-moveX: int
-moveCount: int
-movingup: boolean
-movingdown: boolean
-movingleft: boolean
-movingright: boolean
-direction: int
-imageUp: BufferedImage
-imageUpMoving: BufferedImage
-imageDown: BufferedImage
-imageDownMoving: BufferedImage
-imageLeft: BufferedImage
-imageLeftMoving: BufferedImage
-imageRight: BufferedImage
-imageRightMoving: BufferedImage
-imageWin: BufferedImage
+Frog(int,int,int)
+update():void
+draw(Graphics2D):void
+isWin():boolean
+setWin(boolean):void
+equals(Frog):boolean
+moveOnWater(int):void
+getLife():int
+setLife():void
+increaseLife():void
+decreaseLife():void
+isOnWater():boolean
+setOnWater(boolean):void
+setOnOther(boolean):void
ConfigulableOption
+WEST: int
+NORTH: int
+EAST: int
+SOUTH: int
+LEVEL: int
+SCREEN_WIDTH: int
+SCREEN_HIGHT: int
+ConfigulableOption()
InputUtility
-keyPressed: boolean[]
-keyTriggered: boolean[]
+InputUtility()
+setKeyPressed(int,boolean):void
+setKeyTriggered(int,boolean):void
+getKeyPressed(int):boolean
+getKeyTriggered(int):boolean
+postUpdate():void
Time
-centiSec: int
-second: int
-minute: int
-work: boolean
+Time()
+Time(int,int,int)
+getCentiSec():int
+setCentiSec(int):void
+getSecond():int
+setSecond(int):void
+getMinute():int
+setMinute(int):void
+resetT():void
+isWork():boolean
+setWork(boolean):void
+run():void
GameLogic
-renderableContainer: RenderManager
-gameObjectContainer: List<Entity>
-frog: Frog
+isPause: boolean
+winCount: int
+GameLogic(RenderManager)
#addNewObject(Entity):void
+logicUpdate():void
-newFrog():void

package render



1. Class “logic.Entity”

This class is an abstract class, represents all objects in the game.

1.1 Field

- int **x, y, z**; The current position of object, the coordinate (x, y) is the position of top left corner of object and z is an index of object in z-axis. The more value is on the smaller on the screen.
- int **width, high**; The number of width and high of object.
- boolean **visible**; The status of entity that still visible on screen.
- boolean **destroyed**; the status of entity that be destroyed yet.

1.2 Constructor

- **Entity**(int x, int y, int width, int high, int z); It initializes x, y, width, high, and z then set visible = true and destroyed = false.

1.3 Method

- boolean **isSamePosition**(Entity other); This method checks whether or not the entity and the other position are overlaps.
- abstract void **update**();
- getters of x, y, z, destroyed, and visible
- setters of destroyed and visible

2. Class “logic.Car”

This class is an abstract class, represents a car that extends entity

2.1 Field

- int **vCount**; The number of ticks waiting before moving in each update

  vCout = ConfigulableOption.LEVEL

2.2Constructor

- **Car**(int x, int y, int width, int hight, int z); It initializes all fields.

3. Class "logic.CarBlue"

This class represents a blue car, that extends Car.

3.1 Field

- static BufferedImage **image**; Image of this object, use method Gui.getImage

3.2 Constructor

- **CarBlue**(int x, int y, int z); It initializes all superclass fields. The width and high values get form image.

3.3 Method

- void **update**();

  Step1: Update the value of vCout

  Step2: Move this car to the right 2 pixel if the number of ticks is complete, if this car out of the screen, set its position to the left

- void **draw**(Graphics2D g2); draw the image in position (x, y)

4. Class "logic.CarRed"

This class represents a red car that extends Car.

4.1 Field

- static BufferedImage **image**; Image of this object

4.2 Constructor

- **CarRed**(int x, int y, int z); It initializes all superclass fields. The width and high value get form image.

4.3 Method

- void **update**(); This method is the same as CarBlue
- void **draw**(Graphics2D g2); draw the image in position (x, y)

5. Class "logic.CarPolice"

This class represents a police car that extends Car

5.1 Field

- static BufferedImage **image**; Image of this object

5.2 Constructor

- **CarPolice**(int x, int y, int z); It initializes all superclass fields. The width and high values are get form image.

5.3 Method

- void **update**(); This method is the same as CarBlue, but move to the left 2 pixel instead.(vCount = ConfigulableOption.LEVEL +1; CarPolice move slower than CarRED and CarBlue)
- void **draw**(Graphics2D g2); draw the image in position (x, y)

6. Class logic.ObjectOnWater

This class is an abstract class, represents an object on water that extends entity

6.1 Field

- int **vCout**; The same as Car
- int **oldV**; The value of previous moving velocity of this object

6.2 Constructor

- **ObjectOnWater**(int x, int y, int width, int hight, int z); It initializes all fields.

6.3 Method

- Getter and setter of oldV

7. Class "logic.Log3"

This class represents a log on a river that can place 3 frogs on it

7.1 Field

- static BufferedImage **image**; Image of this object

7.2 Constructor

- **Log3**(int x, int y, int z); It initializes all superclass fields. The width and high values are get form image, but width will use image.getWidth() – 48 because when use method isSamePosition to check a frog is on this log, it's prefectly match.

7.3 Method

- void **update**(); This method is the same CarBlue, and set oldV = 2 if that turn have any movement else set oldV = 0
- void **draw**(Graphics2D g2); draw the image in position (x, y)

8. Class “logic.Log4”

This class represents a log on a river that can place 4 frogs on it

8.1 Field

- static BufferedImage **image**; Image of this object

8.2 Constructor

- **Log4**(int x, int y, int z); It initializes all superclass fields same as Log3

8.3 Method

- void **update**(); This method is the same as Log3

  (vCount = ConfigulableOption.LEVEL + 1)

- void **draw**(Graphics2D g2); draw the image in position (x, y)

9. Class “logic.Turtle”

This class represents a turtle on a river that can place only 1 frog on it

9.1 Field

- static BufferedImage **image**; Image of this object

9.2 Constructor

- **Turtle**(int x, int y, int z); It initializes all superclass fields same as Log3

9.3 Method

- void **update**(); This method is the same as Log3 but move to the left and set

  oldV = -2 (vCount = ConfigulableOption.LEVEL + 2)

- void **draw**(Graphics2D g2); draw the image in position (x, y)

10. Class “logic.Block”

This class represents an object on the game that can’t move and walkthrough, extends Entity

10.1 Constructor

- **Block**(int x, int y, int width, int high, int z);); It initializes all superclass fields

11. Class “logic.Frog”

This class represents a frog on the game which is moveable

11.1 Field

- static int **life**; life point of the frog
- static BufferImaged **imageUp**, **imageDown**, **imageLeft**, **imageRight**, **imageUpMoving**, **imageDownMoving**, **imageLeftMoving**, **imageRightMoving**, **imageWin**; The image of frog in each side and image of frog when it’s moving in each side and image when the frog is on goal
- boolean **isOnWater**; The status is true when the frog is on the water.
- boolean **isOnOther**; The status is true when the frog is on the water and on the objectOnWater
- boolean **isWin**; it’s true when the frog can across the river
- boolean **movingUp**, **movingDown**, **movingLeft**, **movingRight**; The status of the frog that still moving in each direction
- int **moveX**, **moveY**; distance in one move

moveX = ConfigulableOption.SCREEN_WIDTH / 10

moveY = ConfigulableOption.SCREEN_HIGHT / 10

- int **moveCount**; counter of moving

11.2 Constructor

- **Frog**(int x, int y, int z); It initializes all superclass fields

11.3 Method

- void **update**();

  step1: Check whether or not the frog is already win. If yes do nothing.

  Step2: Check the frog that on the water, then set Boolean isOnWater

  Step3: If the frog is moving in each direction, update position and moveCout value.

  Step4: If the frog isn't moving, check key listener from InputUtility that the frog has any movement. If yes, set Boolean moving...(direction) = true

  Step5: Check if y == 64 (the frog is on goal), isWin == true;

- void **draw**(Graphics2D g2);

  If the frog on the goal draw imageWin

  Else if the frog is moving, draw moving image in that direction

  Else is the frog isn't move draw image in that direction

- void **moveOnWater**(int v); This method is called when the frog is on a log or a turtle. If the frog isn't move, set x = x + v

12. Class "logic.GameLogic"

This class is a main logic that controls the entire game.

12.1 Field

- RenderManager **renderableContainer**;
- List<Entity> **gameobjectContainer**; List of all objects on the screen
- Frog **frog**;
- boolean **isPause**;
- int **winCout** = 5; Couter of the frog on the goal

12.2 Constructor

- **GameLogic**(RenderManager renderableContainer); It initializes field and add object into the game. There are 1 frog, 2 CarRed, 2 CarBlue, 2 CarPolice, 2 Log4, 2 Log3, 2 Turtle, and 5 Block.

12.3 Method

- **addNewObject** (Entity entity); This method add entity object into gameObjectContainer and renderableContainer
- void **newFrog**(); set destroyed, invisible and decrease life of the frog and then add new frog into the game.
- void **logicUpdate**();
    - If game is pause, do nothing
    - If the frog win(on the goal) decrease winCout and play sound "goal"
    - If winCount == 0, set gameRuning(false)
    - If the frog is same position of Car, Block, or Frog
        1. Play sound "carimpack", "roar", or "onfrog" respectively

2. Set new frog

- If the frog is same position of ObjectOnWater

1. frog.setonWater(false)

2. frog.moveOnWater(object.getOldV);

- If life <= 0 , set gameRunning(false) and getTime().setWork(false)
- If frog is on the water, set new Frog and play sound “waterbubble”
- If frog is win, set new frog

13. Class “logic. ConfigulableOption”

It stores configuration values data of the game

13.1 Field

- static final int **WEST**, **NORTH**, **EAST**, **SOUTH**; The value of each direction. 0, 1, 2, 3 respectively
- static int **LEVEL**;

  3 = easy

  2 = normal

  1 = hard
- static final int **SCREEN_WIDTH, SCREEN_HIGH**; size of game screen (480 * 640)

14. Class “logic.InputUtility”

This class is used for managing the status of Keylistener.

14.1 Field

- static Boolean[] **keyPressed**; status of key pressing

- static Boolean[] **keyTriggered**; another status for key pressing, it is true only 1 trick.

14.2 Method

- Getters and setters
- void **postUpdate**(); update input status after "end of tick"
- void reset(); for reset status of KeyListener after end game.

15. Class "logic.Time"

This thread represents a time in gamescreen , implements Runnable

15.1 Field

- int **centiSec**;
- int **second**;
- int **minute**;
- static Boolean **work**;

15.2 Constructor

- **Time**(int centiSec, int second, int minute); It initializes all field

15.3 Method

- void **run**(); this method update time while game is running, if game isn't running this thread will wait;

16. Class " highscore.HighScoreManager"

The main class to handling high score.

16.1 Fields

- ArrayList<Score> **easy**; ArrayList for keep records in easy level
- ArrayList<Score> **normal**; ArrayList for keep records in normal level
- ArrayList<Score> **hard**; ArrayList for keep records in hard level
- ObjectOutputStream **outputStream** Object for write the files
- ObjectInputStream **inputStream** Object for read the files

16.2 Constructor

- **HighScoreManager**(); Create ArrayList for every level.

16.3 Methods

- ArrayList<Score> **getScores**(int level); Load score from file and sort scores in ArrayList. That sorted scores by level(level 3 = Easy, level 2 = normal, level 1 = hard
- void **sort**(int level) ; Arrange scores of ArrayList in ascending order.
- void **addScore**(String name, Time time, int level); Add score to the arraylist by level.
- void **loadScoreFile**(int level) ; load score files by inputStream if it don't have the file it will try and catch with FileNotFoundException and create a new default file by outputStream.
- void **updateScoreFile**(int level); Update the new score to the file by outputStream.
- String **getHighscoreString**(int level); Return String of every rank and record in ArrayList

17. Class “highscore.Score” implements Serializable

To record score using the time. Use minute , second, centisecond.

17.1 Field

- int **centiSec**;
- int **second**;
- int **minute**;
- String **name**; The name of player that got that records.

17.2 Constructor

the class used it to create object Score and use it in ArrayList<Score>.

- **Score**(String name, int centiSec, int second, int minute);
- **Score**(String name, Time time);

17.3 Method

- Getter&Setter

18. Class “highscore.ScoreComparator” implements Comparator<Score>

Use it for sorting and ArrayList of score in ascending order.

18.1 Method

- int **compare**(Score score1, Score score2); Use it as comparator for comparing the score.

19. Class “render.GameOverPanel” extends JPanel

Use it to create JPanel of gameover screen

19.1 Constructor

- **GameOverPanel**(); Use it to initializing the size of screen and draw the game over screen by @Override paint component. Add MouseListener for go to main menu panel when player clicked. Have to reset InputUtility after end game by using reset(); .

20. Class"render.GameScreen" extends JComponent implements KeyListener,Runnable

This class is responsible for drawing the game screen (JComponent) , use rendermanager,gamelogic and getting input (listener). We use this class as runnable.

20.1 Field

- static BufferedImage **bg**; keep image of background as bufferedimage.
- static BufferedImage **heart**; keep image of heart as bufferedimage.
- RenderManager **renderManager**; for control about render.
- GameLogic **gameLogic**; use to control game logic.

20.2 Constructor

- **GameScreen**(); to set size of screen, set render manager, set logic manager, and add keylistener.

20.3 Method

- void **paintComponent**(Graphics g); Override the default method to draw all image , font . If the game is pause. It will write "PAUSE" on the screen.
- void **run**(); Using this method to run game. We use synchronized and wait for control when we press enter and the game will pause. And this method is using for update game such as repaint, logic update, render update,

InputUtility update. If player game over, it will stop all sound and go to Gameover panel with gameover sound. If player wins, it will stop all sound and go to Gamewin panel with gamewin sound. It will record the score if the player gets time less time from old record. It will show JOptionpane for record the name of player. If player's name is more than 5, it will substring for not have too long name in records.

- void **keyPressed**(KeyEvent e); this method override the default method. Use it for set "PAUSE" when player pressed enter and protected player from hold button.
- void **keyReleased**(KeyEvent e); For reset value when we released the key.
- static void **setGameRuning**(boolean set) For set gameRunning .

21.Class "render.Gui"

This class used to control panel in the game such as change panel.

21.1 Field

static JFrame **window**;

static JPanel **panel**;

static Time **time**; Runnable for create time thread.

static WavPlayer **sound**; For playing sound.

static HighScoreManager **highScores**; For record highs cores.

21.2 Constructor

Gui(); It will create frame , panel , Runnable of time and play the title sounds.

21.3 Method

static void **goToPanel**(JPanel p); Go to panel p by add JPanel to the window panel and set panel visible to be true.

static void **gameOver**(String state); if String equals to "WIN" it will go to win panel. But if String not equals, it mean the player lose, go to lose panel.

static void **runGame**(); Use for run the game. Starting GameScreen thread and Time thread and play background's sound.

static BufferedImage **getImage**(String s); get image with file name by using ClassLoader.

22.Class "render.HighScorePanel" extends JPanel

Panels that show the top 5 high score records in each levels.(easy,normal,hard)

22.1 Field

- static BufferedImage **bg**; To keep image of background as BufferedImage.

22.2 Constructor

- **HighScorePanel**(); It use to create Highscore panel and create button for swap between easy,normal ,hard highscore, and back button.It have mouse listener to check when the cursor is in button, the button change color. And listener for click button.

22.3 Method

- void **push**(String p);Change background picture to the new picture. Use for changing color when cursor is in the button.
- **drawStringLine**(Graphics2D g2, String text, int x, int y); Draw String in paintcomponent with new line when the String have "\n" .

23. Interface "render.IRenderable"

Interface for drawing objects to the game screen.

24.Class "render.MainmenuPanel" extends JPanel

It's the main menu panel. Can click the button on the panel to play or see the records.

24.1 Field

- static BufferedImage **bg**; To keep image of background as BufferedImage.

24.2 Constructor

- **MainmenuPanel**();To create main menu panel and have button to go select panel or high score panel. Using MouseListenner and method push in the same as "HighScorePanel"

25.Class " render.RenderManager"

Use for manage which object will be render and update if the object is destroyed.

25.1 Field

- List<IRenderable> **entities**; List of object that can render.

25.2 Constructor

- **RenderManager**(); Create an arraylist.

25.3 Method

- void **add**(IRenderable entity); Add and object to the arraylist of renderable object.
- void **update**(); Update that which object is destroyed. If it is destroyed, remove it.
- void **drawScreen**(Graphics2D g2);draw all renderable object.

26. Class "render.SelectLevelPanel" extends JPanel

It's a panel that can select the level of the game.

26.1 Field

- static BufferedImage **bg**; To keep image of background as BufferedImage.

26.2 Constructor

- **SelectLevelPanel**();To create the selectLevelPanel, It has button to select level of the game.(easy , normal , hard)
    - Easy level : ConfigulableOption.LEVEL = 3;
    - Normal level : ConfigulableOption.LEVEL = 2;
    - Hard level : ConfigulableOption.LEVEL = 1;
    - If we click in the button. The game will start in the difficult that we selected. And it have back button for back to MainMenu panel.

27. Class “render.WinPanel” extends JPanel

It‘s a panel that show when player win. Click any on panel will go back to MainMenuPanel. Have to reset InputUtility after end game by using reset(); .

27.1 Constructor

- **WinPanel**(); Create a winpanel using paintComponent.

28. Class “WavPlayer”

Using this class to playing the sound and music in the game.

28.1 Field

- ArrayList<Clip> **clip**;
- ArrayList<String> **clipPlayingMode**; mode of playing (“LOOP”, “SINGLE”, “once”)
- boolean **isPause**; Boolean to check that clip is pause.

28.2 Constructor

- **WavPlayer**(); initialize the Arraylist clip , clipPlayingMode,
- **isPause**;

28.3 Method

- void **play**(String filename, String mode); Clear memory and load file using filename and play song by the mode that input(“LOOP”, “SINGLE”, “once”).
- void **pauseAllSound**(); For use when the game is pause. Stop all music.
- void **resumeAllSound**(); For use when resume from pause, Resume all sound that have mode is “LOOP”.
- boolean **isPause**(); For check that now the game is pause or not.
- void **clearAll**(); Clear all arraylist by create new empty arraylist.
- void **clearMemory**(); If the sound don’t running or active, Close and remove from Arraylist clip and Arraylist clipPlayingMode .
- void **stopAllSound**(); Stop playing all sound in Arraylist clip.