Ali Katlabi OT9IO5

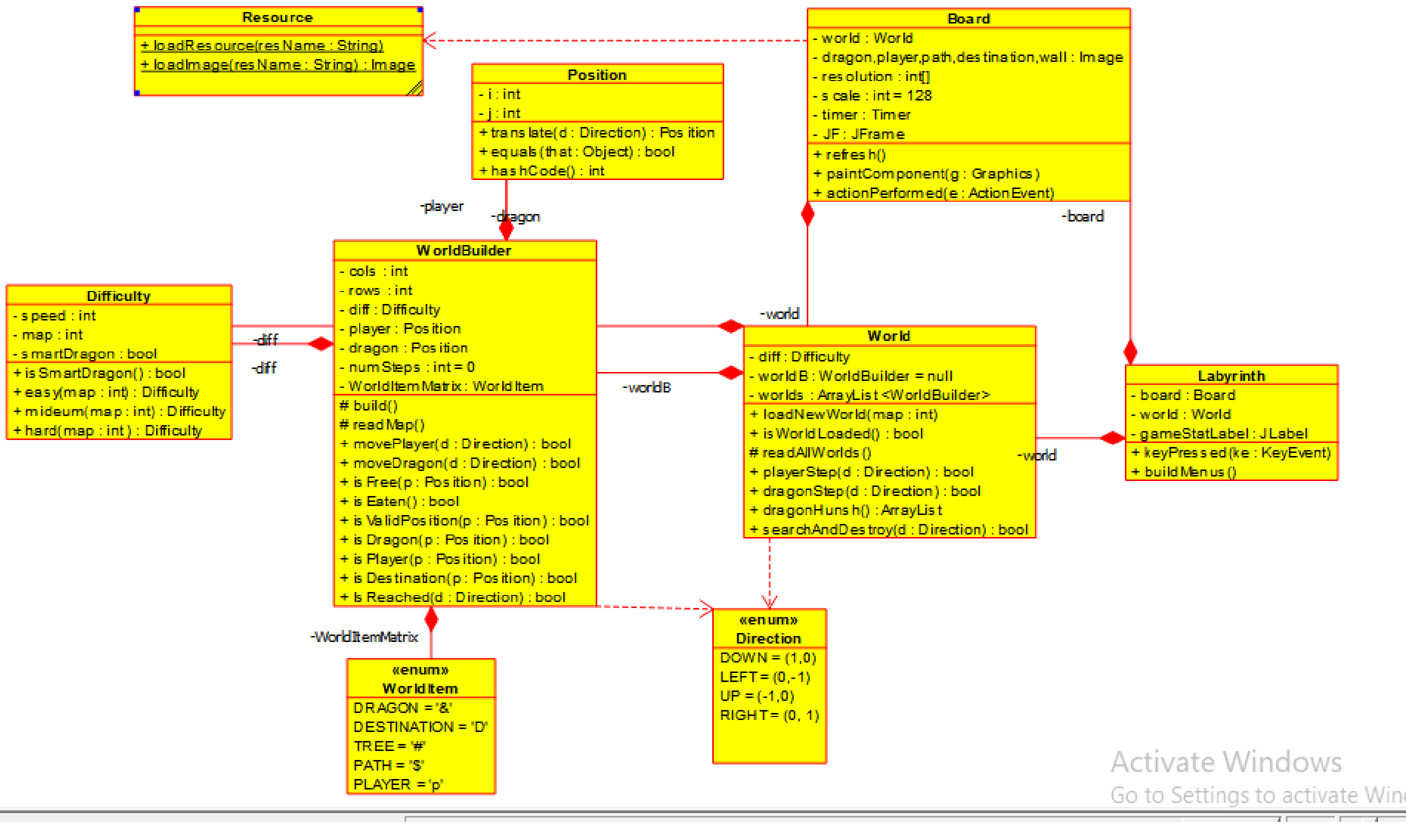
Practical Software Engineering I. Assignement_3

Exercise description:

3. **Labyrinth**
Create the Labyrinth game, where objective of the player is to escape from this labyrinth. The player starts at the bottom left corner of the labyrinth. He has to get to the top right corner of the labyrinth as fast he can, avoiding a meeting with the evil dragon. The player can move only in four directions: left, right, up or down.
There are several escape paths in all labyrinths. The dragon starts off from a randomly chosen position, and moves randomly in the labyrinth so that it choose a direction and goes in that direction until it reaches a wall. Then it chooses randomly a different direction. If the dragon gets to a neighboring field of the player, then the player dies. Because it is dark in the labyrinth, the player can see only the neighboring fields at a distance of 3 units. Record the number of how many labyrinths did the player solve, and if he loses his life, then save this number together with his name into the database. Create a menu item, which displays a highscore table of the players for the 10 best scores. Also, create a menu item which restarts the game.
Take care that the player and the dragon cannot start off on walls.

Ali Katlabi OT9IO5

Class diagram :


Resource
+ loadResource(resName : String)
+ loadImage(resName : String) : Image
Board
- world : World
- dragon,player,path,destination,wall : Image
- resolution : int[]
- scale : int = 128
- timer : Timer
- JF : JFrame
+ refresh()
+ paintComponent(g : Graphics)
+ actionPerformed(e : ActionEvent)
Position
- i : int
- j : int
+ translate(d : Direction) : Position
+ equals(that : Object) : bool
+ hashCode() : int
-player
-dragon
-board
WorldBuilder
- cols : int
- rows : int
- diff : Difficulty
- player : Position
- dragon : Position
- numSteps : int = 0
- WorldItemMatrix : WorldItem
# build()
# readMap()
+ movePlayer(d : Direction) : bool
+ moveDragon(d : Direction) : bool
+ isFree(p : Position) : bool
+ isEaten() : bool
+ isValidPosition(p : Position) : bool
+ isDragon(p : Position) : bool
+ isPlayer(p : Position) : bool
+ isDestination(p : Position) : bool
+ IsReached(d : Direction) : bool
Difficulty
- speed : int
- map : int
- smartDragon : bool
+ isSmartDragon() : bool
+ easy(map : int) : Difficulty
+ mideum(map : int) : Difficulty
+ hard(map : int) : Difficulty
-diff
-diff
-world
-worldB
World
- diff : Difficulty
- worldB : WorldBuilder = null
- worlds : ArrayList <WorldBuilder>
+ loadNewWorld(map : int)
+ isWorldLoaded() : bool
# readAllWorlds()
+ playerStep(d : Direction) : bool
+ dragonStep(d : Direction) : bool
+ dragonHunsh() : ArrayList
+ searchAndDestroy(d : Direction) : bool
-world
Labyrinth
- board : Board
- world : World
- gameStatLabel : JLabel
+ keyPressed(ke : KeyEvent)
+ buildMenus()
-WorldItemMatrix
«enum»
WorldItem
DRAGON = '&'
DESTINATION = 'D'
TREE = '#'
PATH = '$'
PLAYER = 'p'
«enum»
Direction
DOWN = (1,0)
LEFT = (0,-1)
UP = (-1,0)
RIGHT = (0, 1)

Game description:

It is a labyrinth contains basic elements which are the player and the dragon

The purpose is that the player shall find the way out before it got killed by the dragon

description of methods:

class WorldBuilder :

- **void readMap() :** reads the map string for each from a text file contains the map and put it into a String arrayList (stringMap)
- **void build() :** transform the characters of each string map into WorldItem representation so the text map will be represented as matrix of worldItems
- **movePlayer(Direction d) , moveDragon(Direction d) :** manipulate the WorldItems matrix and changes positions to simulate the movments of the elements representing the player and the dragon
- **isValidPosition(Position p):** checks if the some position is out of the limits of the world (worldItems matrix)
- **isPlayer(Position p) , isDragon(Position p):** checks if a certain position of the worldItems matrix is player item or dragon item
- **isDestination(Position p) , IsReached(Direction d):**checking the player reached its way out or not
- **isEaten():** checks if the player item is close to the dragon item
- **isFree(Position p)** :checks If the item of a certain position is a path item so it is free to be substituted with the player item (moving player)

class World :

- **loadNewWorld(int map):** based on the map(level) it will load the corresponding world
- **readAllWorlds():** build the worlds based on the mapID and difficulty , then put them into array of worlds.
- **playerStep(Direction d), dragonStep(Direction d) :** they use movePlayer and moveDragon of the WorldBuilder class
- **dragonHunch() :** return ArrayList of open paths based on searchAndDestroy function
- **searchAndDestroy(Direction dir):** simple algorithm to detect the player position to later inject this detection into dragons steps

class Board:

- **actionPerformed(ActionEvent e):** using this overridden function along with the ticks of the timer we can invoke the repaint() function based on the timer running on the Board
  now we can let the dragon move a randomly but controlled (smartened in maps 2,3,4,5) by the dragonHunch() function .