|  |
| --- |
| Description: https://lh5.googleusercontent.com/ykiT9AqltWwZhr4yy0I8EMArov3nDMgonw73zz3xpW55Pjnk8xUHN8UQ9cSBvnTYiNAypNe7dAXlTfHdscP_Uzi7ewyNNO03_D5Zd9cClbcOOZwZ3EI  CS 319  Object-Oriented Software Engineering Fall 2014  **Conquerors**  API Report    Group H Çağla Arıkan  Metin Enes Uçan  Can Akgün  İsmail Can Dereli |
|  |
|  |
|  |
|  |
|  |

Table of Contents

[Main Menu 1](#_Toc407125475)

[Input Manager 1](#_Toc407125476)

[Point 3](#_Toc407125477)

[Edge 3](#_Toc407125478)

[Edge Manager 4](#_Toc407125479)

[Game Objects 5](#_Toc407125480)

[Hero 5](#_Toc407125481)

[Castle 6](#_Toc407125482)

[Cannonball 7](#_Toc407125483)

[Wall 8](#_Toc407125484)

[Soldiers 8](#_Toc407125485)

[Mine 9](#_Toc407125486)

[Bonus 9](#_Toc407125487)

[Enemies Gone 10](#_Toc407125488)

[Bonus Manager 10](#_Toc407125489)

[Entity Manager 11](#_Toc407125490)

[Conquered Area Detector 12](#_Toc407125491)

[Collision Detector 14](#_Toc407125492)

[Game Over 14](#_Toc407125493)

[High Score File 15](#_Toc407125494)

[Level Selection 16](#_Toc407125495)

[Graphic Engine 16](#_Toc407125496)

[Game Manager 17](#_Toc407125497)

# 

# Main Menu

public class MainMenu {

//atributes

private Rectangle playButton;

private Rectangle viewHelp;

private Rectangle levelSelection;

private Rectangle highScore;

private Rectangle settings;

private Rectangle viewCredits;

private Rectangle exitButton;

//methods

/\* The method below is the constructor of our MainMenu class

\* In this constructor all rectangles for attributes above will

\* be created.

\*/

public MainMenu();

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

/\* This method is for painting the buttons we want to create

\* The font and color will be chosen and rectangles are filled

\* with desired strings. Then draw() method of Java’s graphics

\* class is called for each attribute

\*/

public void paint(Graphics2D g);

}

# Input Manager

public class InputManager implements KeyListener, MouseListener{

//properties

private int velX = 0;

private int velY = 0;

//methods

/\* This method will look which key is pressed. If one of the

\*arrows are chosen then it will update the velX or velY

\* according to the movement’s direction.

\*/

public void keyPressed(KeyEvent e);

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

/\* It will override the keyReleased method of Java’s KeyEvent

\* It will make velX and velY’s values 0.

\*/

public void keyReleased(KeyEvent e);

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

public void keyTyped(KeyEvent arg0);

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

/\* It will check which button is chosen. Then according to the

\* chosen button it will set the state of the game.

\*/

public void mouseClicked(MouseEvent e);

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

public void mouseEntered(MouseEvent arg0);

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

public void mouseExited(MouseEvent arg0);

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

public void mousePressed(MouseEvent arg0);

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

public void mouseReleased(MouseEvent arg0);

}

# Point

public class Point {

//properties

private int x;

private int y;

//methods

/\* The constructor of the class which takes no parameters. It

\* will set point x’s and y^s values as zero.

\*/

Point();

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

// The constructor of the class which takes int x and int y

Point(int x, int y);

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

// Checks if two points are at the same positions.

public boolean equals(Point p);

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

// It finds the distance between two points

Public double getDistance(Point p);

}

# Edge

public class Edge {

//properties

public static final int DIR\_NORTH = 0;

public static final int DIR\_EAST = 1;

public static final int DIR\_SOUTH = 2;

public static final int DIR\_WEST = 3;

private Point start, end;

private int direction;

//methods

// The constructer of the Edge class is given below.

public Edge(Point start, Point end, int direction);

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

/\* This method will check whether the point is on an edge or

\* not

\*/

public boolean isOn(Point point);

}

# Edge Manager

public class EdgeManager {

//properties

private ArrayList<Edge> edgeList; //the array list which

// is mentioned above

//methods

/\* This is the constructor of the EdgeManager. It will

\* create a list which has the 4 edge points of the frame

\*/

public EdgeManager();

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

/\* This method will turn the pointList instances into

\* edges and add them into edge list.

\*/

public void createEdges(ArrayList<Point> pointList);

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

/\*It will check if the movement of the point is to \*

\* somewhere in the conquered area or a free space. With

\* this method we will prevent the point to move inside

\* the conquered area.

\*/

public boolean isPositionValid(Point p);

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

/\* It checks if the point is on edge by looking at the

\* edge list.

\*/

public boolean isOnEdge(Point p);

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

// It checks on which edges the point p stands

public ArrayList<Edge> getStandingEdges(Point p);

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

// It adds an edge to the list

public void addEdge(Edge edge);

}

# Game Objects

public class GameObject {

//properties

private BufferedImage sprite;

private int posX;

private int posY;

private int widht;

private int height;

}

# Hero

public class Hero extends GameObject{

//properties

private int speed;

private int life;

private static Hero h;//for the sake of singleton pattern

private int heroType;

//methods

/\* The constructor of the class. It will set the sprite,

\* speed, its width, height and position and its life(life \* will be one initially). It is private

\* for the sake of singleton pattern.

\*/

private Hero();

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

/\* In this method if a hero is not created before new

\* *hero* is created else the one which exists is return. It

\* is the last step of the singleton pattern.

\*/

public static Hero getHero();

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

/\* It will manage the movement of the hero. While doing

\* that, it will use an instance of edge manager to check

\* if it is on the edge or not or to check if it can move

\* or not

\*/

public void move(int velX, int velY, EdgeManager

eManager);

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

// The life attribute will be increased by one.

public void increaseLife();

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

/\*If life is more than zero the life will be decreased by \* one

\*/

public void decreaseLife();

}

# Castle

public class Castle extends GameObject{

//properties

Private static Castle castle;//singleton method

//method

/\* This is the constructor of the castle. It will set its

\* position.

\*/

private Castle();

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

/\* In this method if a castle is not created before new

\* *castle* is created else the one which exists is return. It

\* is the last step of the singleton pattern.

\*/

public static Hero getCastle();

}

# Cannonball

public class Cannonball extends GameObject {

//properties

private int speed;

private Random rand;

private int walk;

//methods

/\*This is the constructer of the class. This will set the speed

\* and its position.

\*/

public Cannonball();

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

// This will manage then movement of the cannonball.

public void move(EdgeManager eManager);

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

// This will manage then movement of the cannonball backwards.

public void backMove(EdgeManager eManager);

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

}

# Wall

public class Wall extends GameObject {

//properties

private Random rand;

//methods

/\* This is the constructor of the wall class and sets its

\* position.

\*/

public Wall();

}

# Soldiers

public class Soldier extends GameObject{

//properties

private int speed;

private Random rand;

private int direction;

private int walk;

//methods

/\*This is the constructor of the soldier which sets its speed

\* and new Random()

\*/

public Soldier()

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

/\* This will control the movement of the soldier and it will

\* setits position randomly

\*/

public void move(EdgeManager eManager);

}

# Mine

public class Mine extends GameObject {

//properties

private Random rand;

//methods

// The constructor of the mine class

public Mine()

}

# 

# Bonus

public class Bonus {

//properties

private BufferedImage sprite;

private int posX;

private int posY;

private int widht;

private int height;

private boolean powerup;

private boolean badSurprise;

//methods

// This is the constructor of the Bonus

public Bonus();

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

// Returns if bonus is a power-up or not

public boolean isPowerup();

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

// Returns if bonus is a bad-surprise or not

public boolean isBadSurprise();

}

# Enemies Gone

public class EnemiesGone extends Bonus{

//properties

private GameManager gManager;

//methods

/\* This is the constructor of the class. If this bonus is

\* chosen all enemies will be deleted on the screen.

\*/

public EnemiesGone(GameManager gManager);

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

/\* This method overrides the applySpeciality() method of Bonus

\*\* class. This will delete all the enemies in the list.

\*/

public void applySpeciality();

}

# Bonus Manager

public class BonusManager {

//properties

private ArrayList<Bonus> powerups;

private ArrayList<Bonus> badSurprises;

private GameManager gManager;

//methods

/\* This is the constructor of the BonusManager class.

\* This will create instances of powerups and bad-surprises and

\* instantiate gManager and finally call the method

\* fillBonusList().

\*/

public BonusManager(GameManager gManager);

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

/\* This method will randomly create and add bonus objects to

\* the power-ups and bad-Surprises ArrayLists.

\*/

public void fillBonusLists();

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

/\* This method randomly selects and returns a bonus object from

\* the powerups and badSurprises ArrayLists.

\*/

public Bonus selectBonus();

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

/\* This method will check the time of the creation of the

\* bonus on the screen. It informs the GameManager class

\*/

public boolean checkBonusTime();

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

/\* This method will check the time of the destruction of the

\* bonus on the screen. It informs the GameManager class.

\*/

public boolean checkBonusDestructionTime();

}

# Entity Manager

public class EntityManager {

//properties

private ArrayList<GameObject> currentGameObjects;

//methods

/\* This is the constructor of the EntityManager class.

\* it will create a new list of game objects.

\*/

public EntityManager();

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

/\* This will destroy all enemy entities that are in the

\* area under the points in the path ArrayList

\*/

public void destroyEntities(ArrayList<Point> path);

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

/\*This will add the game object (gObject) into the list

\*/

public void addObject(GameObject gObject);

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

/\* This will remove the game object (gObject) from the

\* list

\*/

public void removeObject(GameObject gObject)

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

/\* This will remove the game object by its index from the

\* list

\*/

public void removeObjectByIndex(int index)

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

/\* This will get the object on the given index from the

\* currentGameObjects list.

\*/

public GameObject get(int i)

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

/\* This will return the size of the currentGameObjects

\* list.

\*/

public int size()

}

# Conquered Area Detector

public class ConqueredAreaDetector{

//properties

private Point leave;

private boolean onEdge;

private ArrayList<Point> path;

//methods

/\* This is the constructor of the class. The leave will be set

\* as Point(-1,-1), it will create a new path and onEdge will be \* set as true.

\*/

public ConqueredAreaDetector();

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

//This will get the point on given index from the path list.

public Point get(int i)

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

// This will return the size of the path array list.

public int size()

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

/\* This is the most important method of this class. Since this \* method checks whether an area is conquered or not before the \* every call of the repaint method of GraphicEngine in the \*GameManager. All of the parameters are given above will be \*needed to write this method. It’s main logic is not very \*complex, it firstly checks whether hero’s current position is \*on the one of created edges or not. Then it does specific \*operations according to four possible situations that; \*currently standing on an edge with not standing in the \*previous position or standing in the previous position, and \*currently not standing on an edge with not standing in the \*previous position or standing in the previous position. \*Simply it holds the hero’s positions as a path of Point \*instances and if a conquering occurs, this method invokes \*EdgeManager to create necessary Edge instances according to \*the path, calls EntityManager to destroy game objects that are \*standing on this conquered area and lastly informs \*GraphicEngine with this path in order to add this conquered \*area into the its list.

\*/

public void process(boolean nextOnEdge, Point p, EdgeManager edgeManager, EntityManager entityManager, GraphicEngine gEngine);

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

// This method will find the smallest path that hero drew

public ArrayList<Point> simplifyPath(ArrayList<Point> path);

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

//It will return the points of the complete path that hero drew

public ArrayList<Point> completePath(ArrayList<Point> path,

EdgeManager edgeManager);

}

# Collision Detector

public class CollisionDetector {

// properties

private EntityManager eManager;

// methods

// The constructor of the class

public CollisionDetector(EntityManager eManager)

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

/\* Checks the hero position and detects whether any other game

\* object collides with hero according to their current

\* positions and height/width values. If there is a collision

\* it will decrease life

\*/

public void checkCollision();

}

# Game Over

public class GameOver {

//properties

private GameManager gManager;

private Rectangle gameOver;

private Rectangle scoreRectangle;

private int score;

private boolean calculateScore;

private HighScoreFile highScoreFile;

//methods

// It is the constructor of the class

public GameOver(GameManager gManager);

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

//It overrides the paint method of Java’s AWT class.

public void paint(Graphics2D g);

}

# High Score File

public class HighScoreFile {

//This method reads the high scores from a .txt file

public ArrayList<String> readHighScore();

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

/\*Checks whether the new score is higher than the

\* existing ones

\*/

public boolean isHighScore(int newScore);

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

/\* If the score is a high score(checks it by calling

\* isHighScore(int) method), it will changes the high

\* score .txt file.

\*/

public void changeScore(int newScore);

}

# Level Selection

public class LevelSelection {

//properties

private Rectangle level1Button;

private Rectangle level2Button;

private Rectangle level3Button;

private Rectangle level4Button;

private Rectangle level5Button;

//methods

//The constructor method

public LevelSelection();

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

public void paint(Graphics2D g);

}

# Level Finish

public class LevelFinish {

//properties

private GameManager gManager;

private Rectangle levelFinish;

private Rectangle scoreRectangle;

private Rectangle play;

private int score;

private boolean calculateScore;

//methods

public LevelFinish(GameManager gManager);

public void paint(Graphics2D g);

}

# Game Finish

public class GameFinish {

//properties

private GameManager gManager;

private Rectangle gameFinish;

private Rectangle scoreRectangle;

private int score;

private boolean calculateScore;

//methods

public GameFinish(GameManager gManager);

public void paint(Graphics2D g);

}

# Graphic Engine

public class GraphicEngine extends JPanel{

//properties

private Font font;

private BufferedImage backImage;

private BufferedImage mainBack;

private BufferedImage line;

private BufferedImage creditsBack;

private BufferedImage scoresBack;

private BufferedImage settingsBack;

private BufferedImage levelBack;

private EntityManager eManager;

private InputManager iManager;

private EdgeManager edgeManager;

private ConqueredAreaDetector cAreaDetector;

private long deathTime;

private MainMenu mMenu;

private GameOver gOver;

private ArrayList<Polygon> polygonList;

private GameManager gManager;

public static enum State {PlayGame, MainMenu, ViewHelp,

selectLevel, viewHighScore, Settings, ViewCredits};/\*this

\* is needed since there are different

\* states of the game

\*/

private static State state;

//methods

/\* This is the constructor of the GraphicEngine class.

\* This will create a background image and instantiate

\* all the parameters given. Moreover it will create a

\* frame of the game.

\*/

public GraphicEngine(GameManager gManager, InputManager

iManager, EntityManager eManager, MainMenu mMenu,

ConqueredAreaDetector cAreaDetector);

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

/\* This will override the paint class of the Java’s Graphics

\* class. It will draw the graphics according to given state.

\*/

public void paint(Graphics g);

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

/\* This method is created for adding areas as polygons when

\* an area is conquered. This will traverse the path and

\* two paths will be found after that smallest path will be

\* chosen since this is the drawn path. Then that path create a

\* polygon and added to the polygon ArrayList

\*/

public void addArea(ArrayList<Point> path);

}

# Game Manager

public class GameManager implements ActionListener{

//properties

private int maxTime;

private int level;

private int coefficient;

private Hero hero;

private Castle castle;

private Soldier soldier1;

private Soldier soldier2;

private Soldier soldier3;

private Wall wall1;

private Wall wall2;

private Wall wall3;

private Wall wall4;

private Cannon cannon;

private Cannon reverseCannon;

private Cannonball cannonball;

private Cannonball cannonball2;

private Mine mine;

private InputManager iManager;

private GraphicEngine gEngine;

private EntityManager entityManager;

private EdgeManager edgeManager;

private ConqueredAreaDetector cAreaDetector;

private BonusManager bManager;

private CollisionDetector collisionDetector;

private MainMenu mMenu;

private GameOver gOver;

private Timer timer;

private long lastMove;

private long startTime;

private int remainigTime;

private static boolean *setRemainigTimeBool*;

private static boolean *setRemainingTimeMinus*;

private boolean cannonSprite;

private HighScoreFile highScore;

//methods

/\* This is the constructor class. It will instantiate the

\* properties given above and starts the timer.

\*/

public GameManager();

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

/\* If startTime is -1 then setStartTime() will be called. Else

\* it will check the game is running correctly and all objects

\* are in the states that they should be at. If everything is

\* correct and life of the hero is bigger than zero then it will

\* call gEngine.repaint()and update the time. It will do it

\* every 5 milliseconds.

\* This will provide a smooth graphic.

\*/

public void actionPerformed(ActionEvent arg0);

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

// This method will set startTime as the currentTimeMillis()

public void setStartTime();

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

//This will return the remaining time of the game.

public int getRemainigTime();

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

//This method will convert the time into string.

public String getRemaingTimeString()

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

//This resets the whole game

public void reset();

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

//This will calculate the final score of the player

public int calculateScore();

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

// This will delete all enemies on the game screen

public void deleteAllEnemies();

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

// This will reset the time of the game

public void resetTime();

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

// This method will create new wall instance and add it to the entity list

public void addWalls();

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

// This will increase the hero’s life by one

public void lifePlus();

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

/\* This method will check whether the castle’s land is

\* conquered or not

\*/

public void onObjDestroy(GameObject obj);{

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

// This method will return the high score list

public ArrayList<Integer> getHighScoreList();

}