A-LEVEL PROJECT DOCUMENTATION

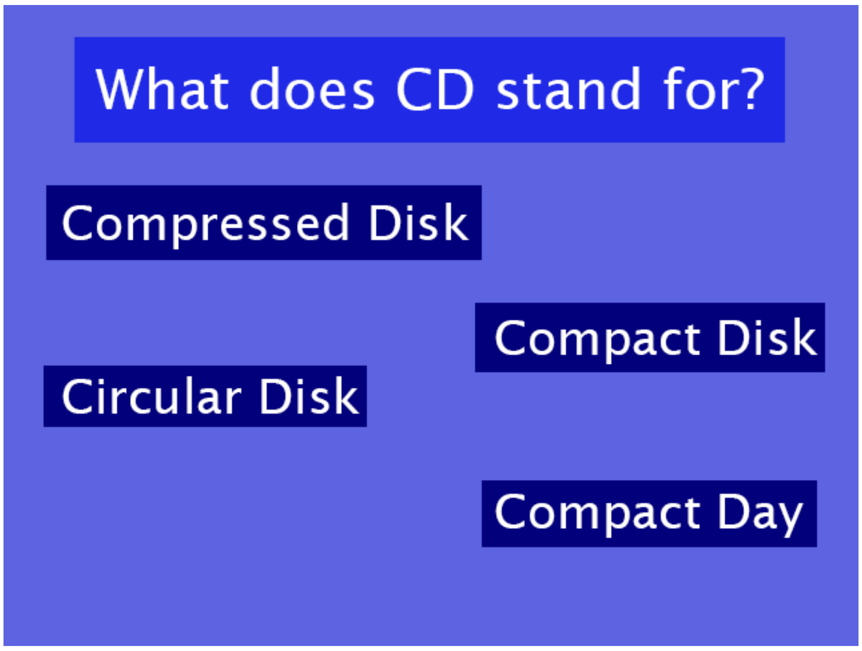
The Maze of Wit

By Edward Sciberras

Contents

Problem Definition

Mr. Sciberras is a teacher that teaches children and students from ages of around eleven years old. He does not teach one specific subject and teaches the children more general knowledge and how to deal with everyday problems. The way he currently is teaching the children is that he has flashcards with questions on them and on the bottom of the cards there are four possible answers. The children must then choose an answer in the hopes of getting it correct. An example of one of these flashcards is shown below.



All the flash cards that Mr. Sciberras is using are all different colours in order to capture the attention of the students better. However, over the past few weeks and months of the students doing the same thing, they started to get bored of it and want to do something new. This is when Mr. Sciberras had the idea of making a game with a similar basis as the flashcards. He then consulted the students to see whether they liked the idea or not. The children agreed and were very excited to play the game when it is finished. Mr. Sciberras had three computers in class but they were old and not that fast. He had a few requirements for how he wanted the game to be.

One of these requirements must be that the flashcards that he uses at school have to be digitalized and put into the game as a fundamental part of it to keep the game educational. This way the children will be having fun and learning at the same time.

Another feature is that the game has to be as captivating as possible in order to engage the students when they are playing it. This will make them want to play the game more and thus learn even more.

The teacher would also like to have as much replayability in the game. This means that the game has to be able to played over and over as much as possible. This has to be done so that the children do not get bored of the game like they did with the flashcards.

In addition, the game also has to include characters and pick-ups so that there will be more content for the children. The pickups can then be scattered along the map of the game.

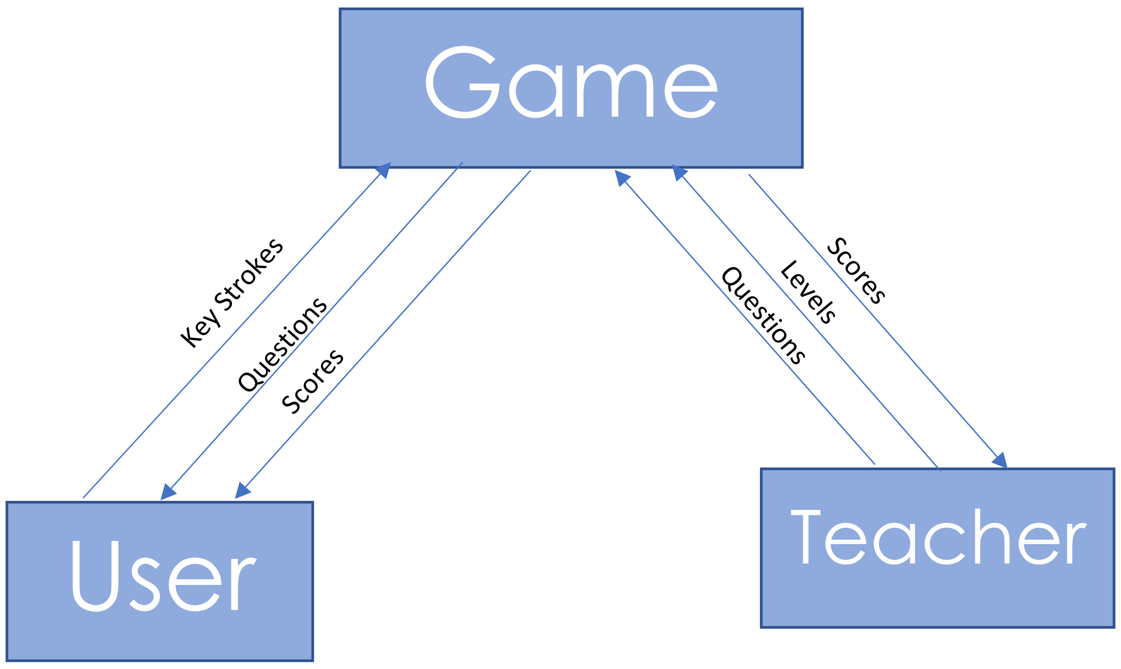
When the game is done it also has to be not graphically intensive so that the old computers in Mr. Sciberras's class room would be able to handle and run the game stably without it crashing.

Mr. Sciberras would also like to be able to add questions easily to the game and have them show up in the game instantly and randomly. This will give him a way to keep the game interesting over time.

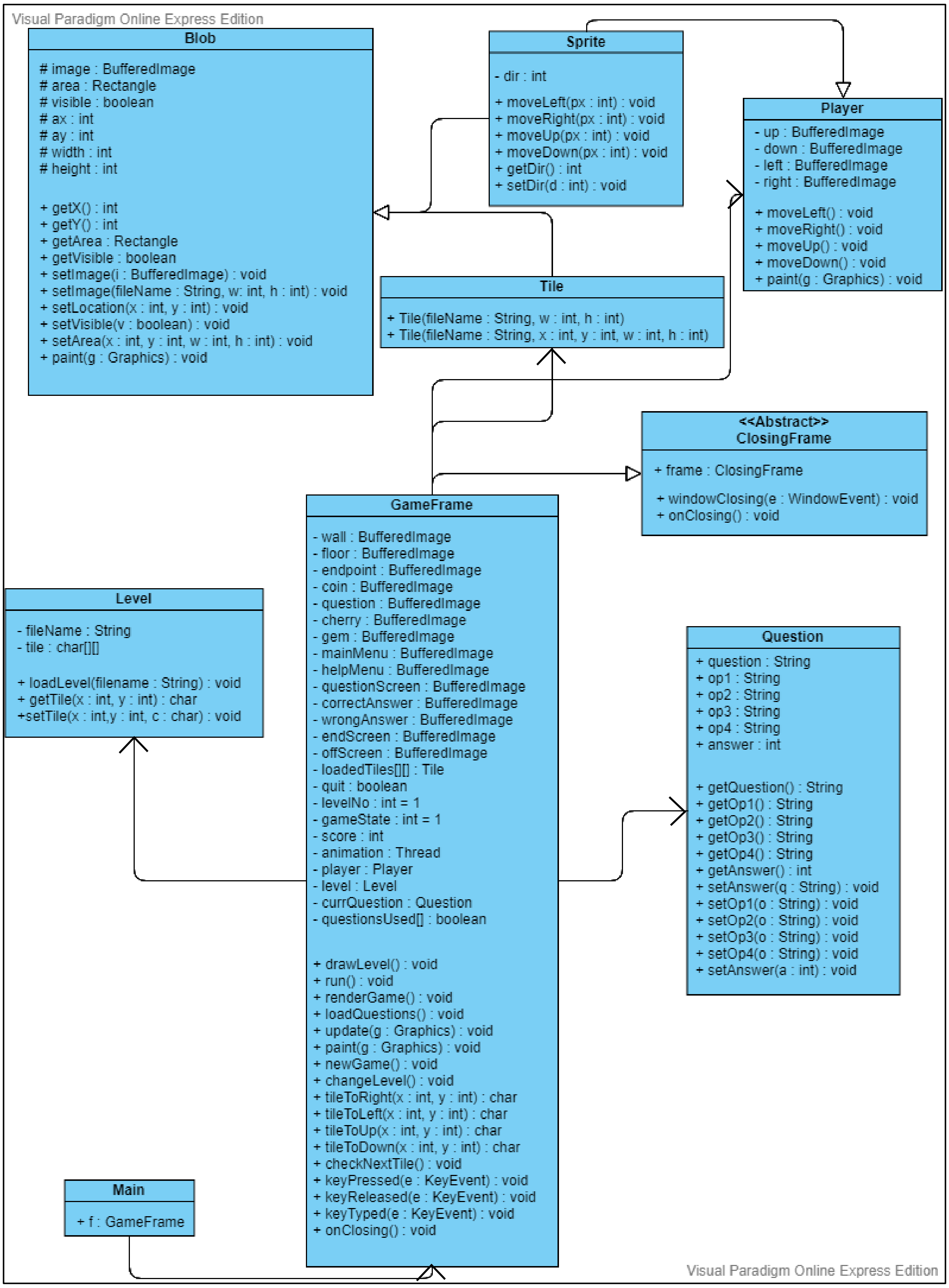
Program Design

2.1 – Data Flow Diagram

Level 0



2.2 – Class Diagram


Visual Paradigm Online Express Edition
Blob
# image : BufferedImage
# area : Rectangle
# visible : boolean
# ax : int
# ay : int
# width : int
# height : int
+ getX() : int
+ getY() : int
+ getArea : Rectangle
+ getVisible : boolean
+ setImage(i : BufferedImage) : void
+ setImage(fileName : String, w: int, h : int) : void
+ setLocation(x : int, y : int) : void
+ setVisible(v : boolean) : void
+ setArea(x : int, y : int, w : int, h : int) : void
+ paint(g : Graphics) : void
Sprite
- dir : int
+ moveLeft(px : int) : void
+ moveRight(px : int) : void
+ moveUp(px : int) : void
+ moveDown(px : int) : void
+ getDir() : int
+ setDir(d : int) : void
Player
- up : BufferedImage
- down : BufferedImage
- left : BufferedImage
- right : BufferedImage
+ moveLeft() : void
+ moveRight() : void
+ moveUp() : void
+ moveDown() : void
+ paint(g : Graphics) : void
Tile
+ Tile(fileName : String, w : int, h : int)
+ Tile(fileName : String, x : int, y : int, w : int, h : int)
<<Abstract>>
ClosingFrame
+ frame : ClosingFrame
+ windowClosing(e : WindowEvent) : void
+ onClosing() : void
GameFrame
- wall : BufferedImage
- floor : BufferedImage
- endpoint : BufferedImage
- coin : BufferedImage
- question : BufferedImage
- cherry : BufferedImage
- gem : BufferedImage
- mainMenu : BufferedImage
- helpMenu : BufferedImage
- questionScreen : BufferedImage
- correctAnswer : BufferedImage
- wrongAnswer : BufferedImage
- endScreen : BufferedImage
- offScreen : BufferedImage
- loadedTiles[][] : Tile
- quit : boolean
- levelNo : int = 1
- gameState : int = 1
- score : int
- animation : Thread
- player : Player
- level : Level
- currQuestion : Question
- questionsUsed[] : boolean
+ drawLevel() : void
+ run() : void
+ renderGame() : void
+ loadQuestions() : void
+ update(g : Graphics) : void
+ paint(g : Graphics) : void
+ newGame() : void
+ changeLevel() : void
+ tileToRight(x : int, y : int) : char
+ tileToLeft(x : int, y : int) : char
+ tileToUp(x : int, y : int) : char
+ tileToDown(x : int, y : int) : char
+ checkNextTile() : void
+ keyPressed(e : KeyEvent) : void
+ keyReleased(e : KeyEvent) : void
+ keyTyped(e : KeyEvent) : void
+ onClosing() : void
Level
- fileName : String
- tile : char[][]
+ loadLevel(filename : String) : void
+ getTile(x : int, y : int) : char
+setTile(x : int,y : int, c : char) : void
Question
+ question : String
+ op1 : String
+ op2 : String
+ op3 : String
+ op4 : String
+ answer : int
+ getQuestion() : String
+ getOp1() : String
+ getOp2() : String
+ getOp3() : String
+ getOp4() : String
+ getAnswer() : int
+ setAnswer(q : String) : void
+ setOp1(o : String) : void
+ setOp2(o : String) : void
+ setOp3(o : String) : void
+ setOp4(o : String) : void
+ setAnswer(a : int) : void
Main
+ f : GameFrame
Visual Paradigm Online Express Edition

2.3 - Flowcharts and Pseudo-Code

Blob:

Variables:

Name	Type	Access Modifier	Description
image	BufferedImage	protected	To pass sprite image
area	Rectangle	protected	To find x and y values
visible	boolean	protected	To see if sprite is showing or not
ax	int	protected	X value
ay	int	protected	Y value
width	int	protected	Width value
height	int	protected	Height value

Methods:

Name	Type	Access Modifier	Parameters	Description
Blob	N/A	public	String fileName int w, h	Constructor
Blob	N/A	public	String fileName int x, y, w, h	Constructor
getX	int	public	N/A	To get the value of X
getY	int	public	N/A	To get the value of Y
getArea	Rectangle	public	N/A	To get the area

getVisible	boolean	public	N/A	To get the data of Visible
setImage	void	public	BufferedImage i	To set an image
setImage	void	public	String fileName int w, h	To set an image
setLocation	void	public	int x, y	To change the location
setVisible	void	public	boolean v	To change the visibility
setArea	void	public	int x, y, w, h	To set the area of an item
paint	void	public	Graphics g	To paint the image on screen

Sprite

Variables:

Name	**Type**	**Access Modifier**	**Description**
dir	int	private	To determine the direction of the player

Methods:

Name	Type	Access Modifier	Parameters	Description
Sprite	N/A	public	String fileName int w, h	Constructor
Sprite	N/A	public	String fileName int x, y, w, h	Constructor
moveLeft	void	public	int px	To move left
moveRight	void	public	int px	To move right
moveUp	void	public	int px	To move up
moveDown	void	public	int px	To move down
getDir	int	public	N/A	To get the value of the direction
setDir	void	public	int d	To set the direction

Player

Variables:

Name	Type	Access Modifier	Description
up	BufferedImage	private	Storing the image of the up sprite
down	BufferedImage	private	Storing the image of the down sprite
left	BufferedImage	private	Storing the image of the left sprite
right	BufferedImage	private	Storing the image of the right sprite

Methods:

Name	Type	Access Modifier	Parameters	Description
Player	N/A	public	String fileName int x, y, w, h	Constructor
moveLeft	void	public	N/A	To move left
moveRight	void	public	N/A	To move right
moveUp	void	public	N/A	To move up
moveDown	void	public	N/A	To move down
paint	void	public	Graphics g	To paint the player on screen

Tile

Methods:

Name	Type	Access Modifier	Parameters	Description
Tile	N/A	public	String fileName int w, h	Constructor
Tile	N/A	public	String fileName int x, y, w, h	Constructor

Question:

Variables:

Name	Type	Access Modifier	Description
question	String	private	Stores the actual question
op1	int	private	Stores the first option of the question
op2	int	private	Stores the second option of the question
op3	int	private	Stores the third option of the question
op4	int	private	Stores the fourth option of the question
answer	int	private	Stores the numerical answer of the question

Methods:

Name	Type	Access Modifier	Parameters	Description
getQuestion	String	public	N/A	Get actual question
getOp1	String	public	N/A	Get first option of question
getOp2	String	public	N/A	Get second option of question
getOp3	String	public	N/A	Get third option of question
getOp4	String	public	N/A	Get fourth option of question
getAnswer	int	public	N/A	Get answer for the question
setQuestion	void	public	String q	Set actual question
setOp1	void	public	String o	Set first option of question
setOp2	void	public	String o	Set second option of question
setOp3	void	public	String o	Set third option of question
setOp4	void	public	String o	Set fourth option of question
setAnswer	void	public	int a	Set answer for the question

Level

Variables:

Name	Type	Access Modifier	Description
fileName	String	private	Stores the fileName
tile[][]	char	private	Stores the tiles in the order of the level

Methods:

Name	Type	Access Modifier	Parameters	Description
getTile	char	public	int x, y	Gets tile value
setTile	void	public	int x, y char c	Changes the given tile

ClosingFrame

Methods:

Name	Type	Access Modifier	Parameters	Description
ClosingFrame	N/A	public	String title int width, height	Constructor
onClosing	void	public	N/A	Meant to be overridden to close the window
MyWindowAdapter	N/A	public	ClosingFrame f	Creates a pointer to ClosingFrame
windowClosing	void	public	WindowEvent e	Closes the window

GameFrame:

Variables:

Name	Type	Access Modifier	Description
wall	BufferedImage	private	Stores the image of the wall
floor	BufferedImage	private	Stores the image of the floor
endpoint	BufferedImage	private	Stores the image of the end point
coin	BufferedImage	private	Stores the image of coin
question	BufferedImage	private	Stores the image of question
cherry	BufferedImage	private	Stores the image of cherry
gem	BufferedImage	private	Stores the image of gem
mainMenu	BufferedImage	private	Stores the image of Main Menu
helpMenu	BufferedImage	private	Stores the image of Help Menu
questionScreen	BufferedImage	private	Stores the image of Question Screen
correctAnswer	BufferedImage	private	Stores the image of Correct Screen
wrongAnswer	BufferedImage	private	Stores the image of Wrong Screen
endScreen	BufferedImage	private	Stores the image of End Screen
offScreen	BufferedImage	private	Used to paint the image before it goes on screen to make the image smoother
loadedTiles[][]	Tile	private	Stores the tiles before they are painted on the screen

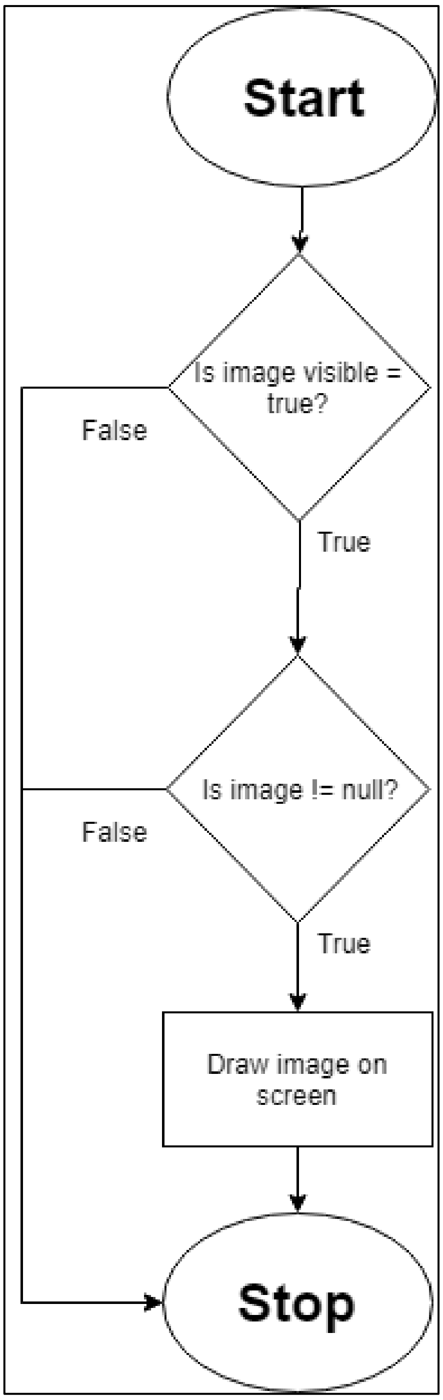
quit	boolean	private	Checks if the user wants to quit or not and closes the window
levelNo	int	private	Stores the current level number
score	int	private	Stores the score of the player
gameState	int	private	Store the current state that the game is in
animation	Thread	private	Thread for animation to make it easier to handle
player	Player	private	The player
level	Level	private	The level
currQuestion	Question	private	The current question
questionsUsed[]	boolean	private	An array to store which questions were used and asked

<u>Methods:</u>

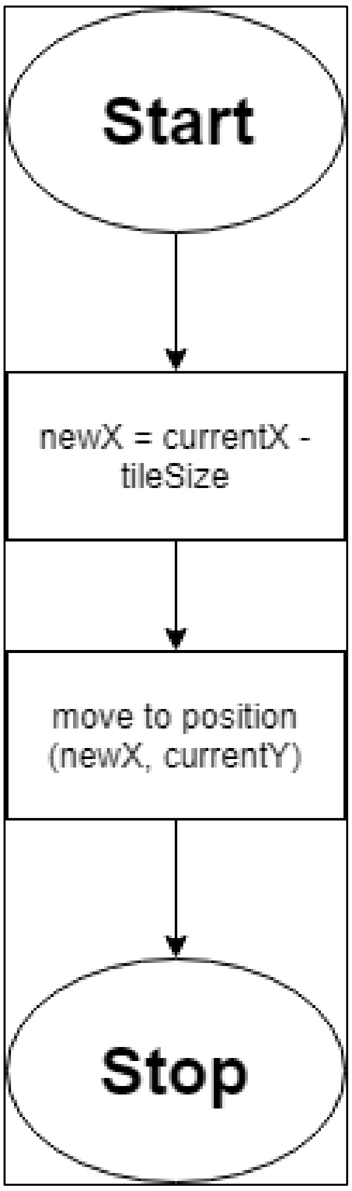
Name	Type	Access Modifier	Parameters	Description
GameFrame	N/A	public	String t int w, h	Constructor
drawLevel	void	public	N/A	Paints the level on the screen
run	void	public	N/A	Runs the game and controls thread
renderGame	void	public	N/A	Renders game with different game states
loadQuestions	void	public	N/A	Loads the questions into the arraylist
update	void	public	Graphics g	Calls the pain

paint	void	public	Graphics g	Paints the image on the offScreen
newGame	void	public	N/A	Resets the game for a new playthrough
changeLevel	void	public	N/A	Changes the level when the player reaches the end
tileToRight	char	public	int x, y	Checks the next tile to the right
tileToLeft	char	public	int x, y	Checks the next tile to the left
tileToDown	char	public	int x, y	Checks the next tile below it
tileToUp	char	public	int x, y	Checks the next tile above it
checkNexTile	void	public	N/A	Check the next tile for items
keyPressed	void	public	KeyEvent e	Checks which key was pressed
keyReleased	void	public	KeyEvent e	Not used
keyTyped	void	public	KeyEvent e	Not used
onClosing	void	public	N/A	Closes the window

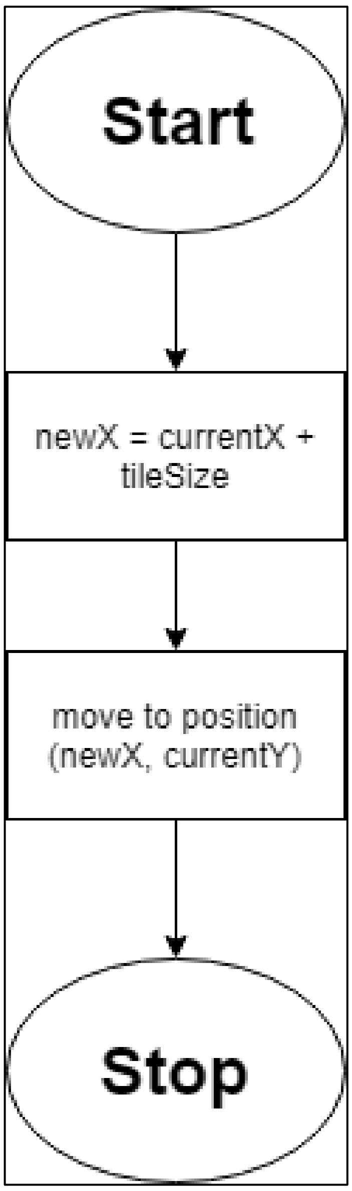
Blob – paint()


Start
Is image visible = true?
False
True
Is image != null?
False
True
Draw image on screen
Stop

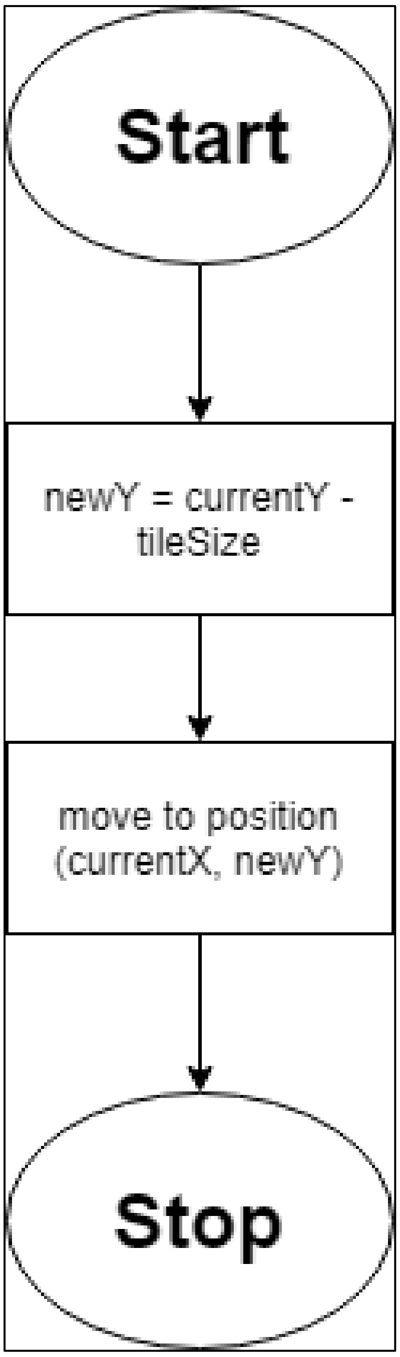
<u>Sprite – moveLeft()</u>



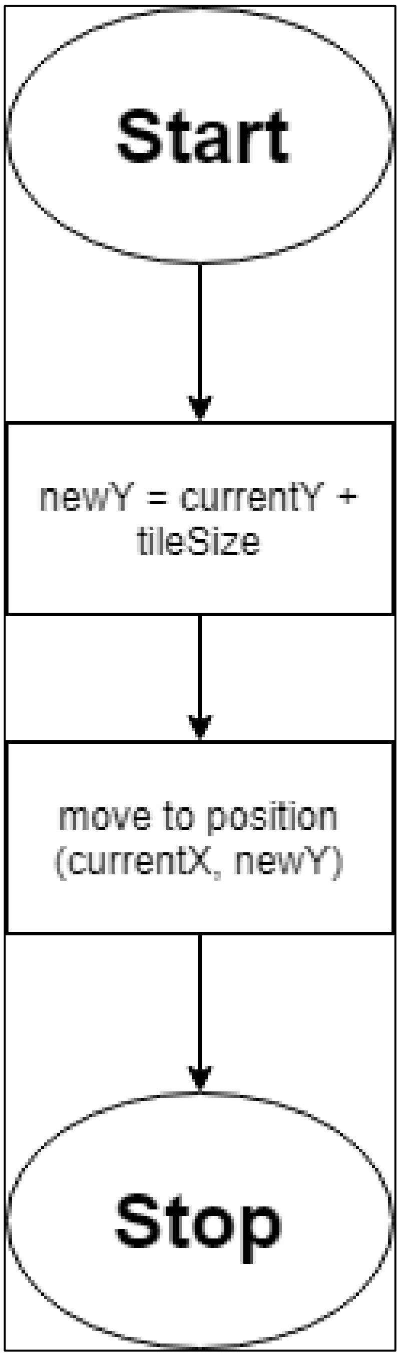
<u>Sprite – moveRight()</u>



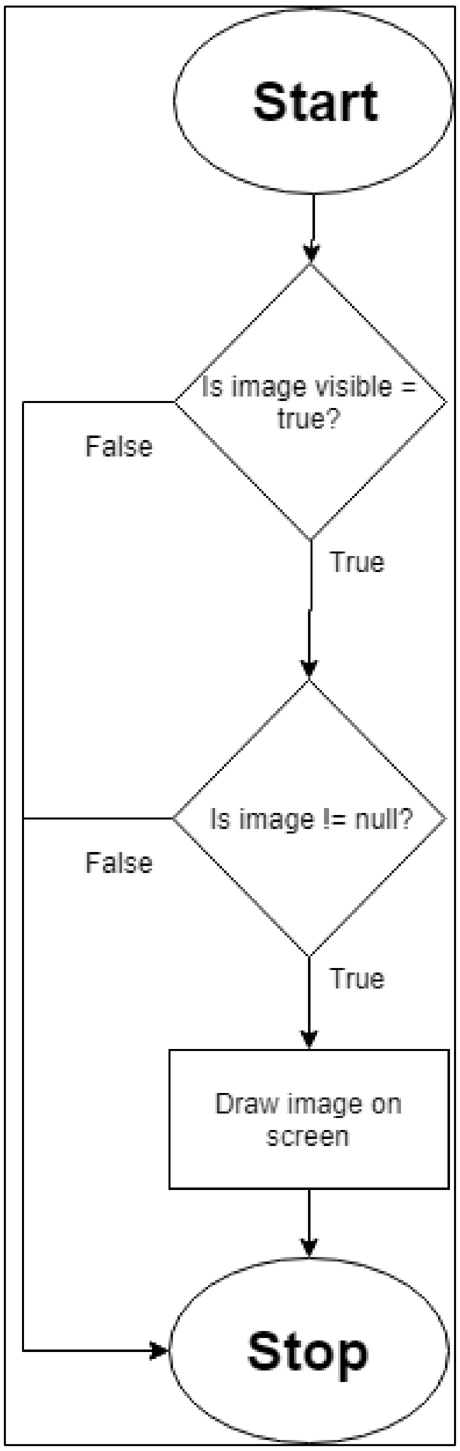
Sprire – moveUp()



Sprite- moveDown()



Player - paint()



Level – loadLevel()

```
tile(15, 20)
for i from 0 to 15
    move to line i
    for j from 0 to 20
        tile(i, j) = char from text file
end
```

GameFrame – GameFrame()

```
if no error
    load images of sprites
else
    display error message
if no error
    load images of menus
else
    display error message
quit = false
offScreen = blank image size of screen
loadLevel(Maze1)
loadQuestions()
start animation Thread
new player facing down
player in starting position
```

GameFrame – drawLevel()

```
for i from 0 to 15
    for j from 0 to 20
        getTile(j, i)
            if tile == wall
                draw wall
            if tile == floor
                draw floor
            if tile == endpoint
                draw endpoint
            if tile == coin
                draw coin
            if tile == question
                draw question
            if tile == cherry
                draw cherry
            if tile == gem
                draw gem
            x = x + 32
    y = y + 32
    x = 0
end
```

GameFrame – run()

```
while quit = false
    renderGame()
    repaint()
    if no error
        Make thread sleep for 30 ms
    else
        Display error message
```

GameFrame – renderGame()

```
if gameState = MAIN_MENU
    display mainMenu image
if gameState = PLAYING_GAME
    draw level
    draw player
    display score
if gameState = HELP_MENU
    display helpMenu image
if gameState = END_SCREEN
    display endScreen image
    show final score
if gameState = QUESTION_SCREEN
    display question image
    do
        ran = random number from 0 to number of questions
    while(question was not already asked)
    draw question on screen
    draw options on screen
    gameState = ANSWER_SCREEN
if gameState = ANSWER_SCREEN
    wait for user to input answer
if gameState = CORRECT_ANSWER
    display correctAnswer image
if gameState = WRONG_ANSWER
    display wrongAnswer image
```

GameFrame – loadQuestions()

```
if no error
    open text tile “questions.txt”
    while(next line is not empty)
        q = new question
        q.setQuestion(nextLine)
        q.setOption1(nextLine)
        q.setOption2(nextLine)
        q.setOption3(nextLine)
        q.setOption4(nextLine)
        q.setAnswer(nextNumber)
    add q to arrayList of questions
    questionsUsed[] = boolean array of size questions
    for each question in questionsUsed
        question = false
else
    display error message
```

GameFrame – newGame()

```
score = 0
loadLevel(Maze1)
set players location to starting point
```

GameFrame – changeLevel()

```
if levelNo = 1
    loadLevel(Maze2)
    levelNo++
    set players location to starting point
else if levelNo = 2
    loadLevel(Maze3)
    levelNo++
    set players location to starting point
else if levelNo = 3
    gameState = END_SCREEN
```

GameFrame – tileToRight()

```
posX = currentX/32 + 1
posY = currentY/32
if posX < 20
    return tile(posX, posY)
else
    return FLOOR
```

GameFrame – tileToLeft()

```
posX = currentX/32 - 1
posY = currentY/32
if posX < 20
    return tile(posX, posY)
else
    return FLOOR
```

GameFrame – tileToDown()

```
posX = currentX/32
posY = currentY/32 + 1
if posX < 15
        return tile(posX, posY)
else
        return FLOOR
```

GameFrame – tileToUp()

```
posX = currentX/32
posY = currentY/32 - 1
if posX < 15
        return tile(posX, posY)
else
        return FLOOR
```

GameFrame – checkNextTile()

```
px = currentX
py = currentY
if(getTile = COIN or CHERRY or GEM)
        change tile to FLOOR
        score++
else if(getTile = ENDPOINT)
        changeLevel()
else if(getTile = QUESTION)
        gameState = QUESTION_SCREEN
        change tile to FLOOR
```

GameFrame – keyPressed()

```
px = currentX
py = currentY
if keypress = LEFT or A
    if tileToLeft != WALL
        player.moveLeft
        px = currentX
        py = currentY
        checkNextTile()

if keypress = RIGHT or D
    if tileToRight != WALL
        player.moveRight
        px = currentX
        py = currentY
        checkNextTile()

if keypress = DOWN or S
    if tileToDown != WALL
        player.moveDown
        px = currentX
        py = currentY
        checkNextTile()

if keypress = UP or W
    if tileToUp != WALL
        player.moveUp
        px = currentX
        py = currentY
        checkNextTile()
```

```
if keypress = F1
        if gameState = MAIN_MENU
                newGame()
                gameState = PLAYING_GAME
        if gameState = HELP_MENU
                gameState = MAIN_MENU
        if gameState = END_SCREEN
                gameState = MAIN_MENU

if keypress = F2
        if gameState = MANU_MENU
                gamestate = HELP_MENU

if keypress = F3
        if gameState = MAIN_MENU
                quit = true
                exit game

if keypress = 1
        if gameState = ANSWER_SCREEN
                if 1 = currentQuestion answer
                        score = score + 2
                        gameState = CORRECT_ANSWER
                else
                        gameState = WRONG_ANSWER

if keypress = 2
        if gameState = ANSWER_SCREEN
                if 2 = currentQuestion answer
                        score = score + 2
                        gameState = CORRECT_ANSWER
                else
                        gameState = WRONG_ANSWER
```

```
if keypress = 3
    if gameState = ANSWER_SCREEN
        if 3 = currentQuestion answer
            score = score + 2
            gameState = CORRECT_ANSWER
        else
            gameState = WRONG_ANSWER


if keypress = 4
    if gameState = ANSWER_SCREEN
        if 4 = currentQuestion answer
            score = score + 2
            gameState = CORRECT_ANSWER
        else
            gameState = WRONG_ANSWER

if keypress = Spacebar
    if gameState = CORRECT_ANSWER or WRONG_ANSWER
        gameState = PLAYING_GAME
```

GameFrame – onClosing()

```
quit = true;
exit game
```

Program Listing

3.1 - Printout of the Program

Blob:

```
import java.awt.image.*;
import javax.imageio.*;
import java.awt.*;
import java.io.*;
import javax.swing.*;
public class Blob{
    protected BufferedImage image;
    protected Rectangle area;
    protected boolean visible;
    protected int ax, ay, width, height;

    // Contructor with parameters of fileName, width and height
    public Blob(String fileName, int w, int h){
        try{
            image = ImageIO.read(new File(fileName));
            area = new Rectangle(0, 0, w, h);
            //visible = false;
        } catch (Exception e){
            JOptionPane.showMessageDialog(null, "Error in Blob
constructor\nError Message: " + e.getMessage(), "Blob Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    // Contructor with parameters of fileName, x value, y value, width and
height
    public Blob(String fileName, int x, int y, int w, int h){
        try{
            image = ImageIO.read(new File(fileName));
            area = new Rectangle(x, y, w, h);
            //visible = false;
        } catch (Exception e){
            JOptionPane.showMessageDialog(null, "Error in Blob
constructor\nError Message: " + e.getMessage(), "Blob Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    // Getters
    public int getX(){
        return (int)area.getX();
    }

    public int getY(){
        return (int)area.getY();
    }

    public Rectangle getArea(){
        return area;
    }

    public boolean getVisible(){
        return visible;
    }

    // Setters
    public void setImage(BufferedImage i){
        image = i;
    }

    public void setImage(String fileName, int w, int h){
        try{
            image = ImageIO.read(new File(fileName));
            area = new Rectangle(0, 0, w, h);
        } catch (Exception e){
            JOptionPane.showMessageDialog(null, "Error setting the 
Image\nError Message: " + e.getMessage(), "Image Setting Error", 
JOptionPane.ERROR_MESSAGE);
        }
    }

    public void setLocation(int x, int y){
        area.setLocation(x, y);
    }

    public void setVisible(boolean v){
        visible = v;
    }

    public void setArea(int x, int y, int w, int h){
        ax = x;
        ay = y;
        width = w;
        height = h;
    }

    // Draws the image
    public void paint(Graphics g){
        if(visible && image != null){
            g.drawImage(image, (int)area.getX(), (int)area.getY(), null);
        }
    }
}
```

Sprite:

```
public class Sprite extends Blob{
    private int dir;

    // Contructor with parameters of fileName, width and height
    public Sprite(String fileName, int w, int h){
        super(fileName, w, h);
    }

    // Contructor with parameters of fileName, x value, y value, width and 
height
    public Sprite(String fileName, int x, int y, int w, int h){
        super(fileName, x, y, w, h);
    }

    // Move Controls
    public void moveLeft(int px){
        int newX = (int)getX() - px;
        setLocation(newX, (int)getY());
    }
    public void moveRight(int px){
        int newX = (int)getX() + px;
        setLocation(newX, (int)getY());
    }

    public void moveUp(int px){
        int newY = (int)getY() - px;
        setLocation((int)getX(), newY);
    }

    public void moveDown(int px){
        int newY = (int)getY() + px;
        setLocation((int)getX(), newY);
    }

    // Getters
    public int getDir(){
        return dir;
    }

    // Setters
    public void setDir(int d){
        dir = d;
    }
}
```

Player:

```
import java.io.*;
import java.awt.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;
public class Player extends Sprite{
    private BufferedImage up, down, left, right;

    private final String FACING_UP = "Sprites/Facing-Up.png";
    private final String FACING_DOWN = "Sprites/Facing-Down.png";
    private final String FACING_LEFT = "Sprites/Facing-Left.png";
    private final String FACING_RIGHT = "Sprites/Facing-Right.png";

    public Player(String fileName, int x, int y, int w, int h){
        super(fileName, x, y, w, h);

        //loading in sprites for player
        try {
            up = ImageIO.read(new File(FACING_UP));
            down = ImageIO.read(new File(FACING_DOWN));
            left = ImageIO.read(new File(FACING_LEFT));
            right = ImageIO.read(new File(FACING_RIGHT));
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error loading sprites of
character\nError Message: " + e.getMessage(), "Character Sprites Loading
Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    public void moveUp(){
        super.moveUp(32);

        image = up;
    }

    public void moveRight(){
        super.moveRight(32);

        image = right;
    }

    public void moveDown(){
        super.moveDown(32);

        image = down;
    }

    public void moveLeft(){
        super.moveLeft(32);

        image = left;
    }

    @Override
    public void paint(Graphics g){
        if(visible && image != null){
            g.drawImage(image, (int)area.getX(), (int)area.getY(), null);
        }
    }
}
```

Tile:

```
import java.io.*;
import javax.imageio.*;
import java.awt.image.*;
import javax.swing.*;
public class Tile extends Blob{
    public Tile(String fileName, int w, int h){
        super(fileName, w, h);
    }

    public Tile(String fileName, int x, int y, int w, int h){
        super(fileName, x, y, w, h);
    }
}
```

Question:

```
public class Question{
    String question, op1, op2, op3, op4;
    int answer;

    // Getters
    public String getQuestion(){
        return question;
    }
    public String getOp1(){
        return op1;
    }

    public String getOp2(){
        return op2;
    }

    public String getOp3(){
        return op3;
    }

    public String getOp4(){
        return op4;
    }

    public int getAnswer(){
        return answer;
    }

    // Setters
    public void setQuestion(String q){
        question = q;
    }

    public void setOp1(String o){
        op1 = o;
    }

    public void setOp2(String o){
        op2 = o;
    }

    public void setOp3(String o){
        op3 = o;
    }

    public void setOp4(String o){
        op4 = o;
    }

    public void setAnswer(int a){
        answer = a;
    }
}
```

Level:

```
import java.io.*;
import javax.swing.*;
public class Level{
    private String fileName;

    private char[][] tile; // screen is 480 by 640 so 15 and 20 (tile is 
32x32)

    public void loadLevel(String fileName){
        String line = "";
        try{
            tile = new char[15][20];
            BufferedReader f = new BufferedReader(new 
FileReader(fileName));
            for(int i = 0; i < 15; i++){ // loop for y
                line = f.readLine();
                for(int j = 0; j < 20; j++){ // loop for x
                    tile[i][j] = line.charAt(j); // loading char into array
                    //System.out.print(tile[i][j]);
                }
                //System.out.println();
            }
            //System.out.println("\n\n");
            f.close();
        }
        catch(Exception e){
            JOptionPane.showMessageDialog(null, "Error loading in 
level\nError Message: " + e.getMessage(), "Level Loading Error", 
JOptionPane.ERROR_MESSAGE);
        }
    }

    // Getter
    public char getTile(int x, int y){
        return tile[y][x];
    }

    // Setter
    public void setTile(int x, int y, char c){
        tile[y][x] = c;
    }
}
```

ClosingFrame:

```
import java.awt.*;
import java.awt.event.*;

public abstract class ClosingFrame extends Frame{
    public ClosingFrame(String title, int width, int height){
        super(title);
        setSize(width, height);

        addWindowListener(new MyWindowAdapter(this)); //calling instance

    }
    public abstract void onClosing();
}

class MyWindowAdapter extends WindowAdapter {

    ClosingFrame frame; // creating pointer to above class

    MyWindowAdapter (ClosingFrame f) {
        frame = f;
    }

    @Override
    public void windowClosing(WindowEvent e){
        frame.onClosing();
    }

}
```

GameFrame:

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.imageio.*;
import java.util.*;
import javax.swing.*;
public class GameFrame extends ClosingFrame implements Runnable,
KeyListener{
    private BufferedImage wall, floor, endPoint, coin, question, cherry,
gem; //images for sprites
    private BufferedImage mainMenu, helpMenu, questionScreen,
correctAnswer, wrongAnswer, endScreen; //images for menus and screens
    private BufferedImage offScreen; // drawn before shown to prevent
screen tearing
    private Tile loadedTiles[][]; //to store tiles
    private boolean quit; //see if the user wants to quit or not
    private int levelNo = 1, score, gameState = 1;
    private Thread animation; //thread for animation
    private Player player;
    private Level level;
    private Question currQuestion; //current random question
    private boolean questionsUsed[]; //array for questions that are already
used

    // constants for sprites images
    private final String WALL_SPRITE = "Sprites/wall.png";
    private final String FLOOR_SPRITE = "Sprites/floor.png";
    private final String ENDPOINT_SPRITE = "Sprites/end-point.png";
    private final String COIN_SPRITE = "Sprites/coin.png";
    private final String QUESTION_SPRITE = "Sprites/question.png";
    private final String CHERRY_SPRITE = "Sprites/cherry.png";
    private final String GEM_SPRITE = "Sprites/gem.png";
    private final String FACING_DOWN = "Sprites/Facing-Down.png";

    // constants for menu images
    private final String MAIN_MENU_SCREEN = "ScreensAndMenus/MainMenu.jpg";
    private final String HELP_MENU_SCREEN = "ScreensAndMenus/HelpMenu.jpg";
    private final String QUESTION_TEMPLATE_SCREEN =
"ScreensAndMenus/QuestionTemplate.jpg";
    private final String CORRECT_SCREEN =
"ScreensAndMenus/CorrectScreen.jpg";
    private final String WRONG_SCREEN = "ScreensAndMenus/WrongScreen.jpg";
    private final String END_MENU_SCREEN = "ScreensAndMenus/EndScreen.jpg";

    // constants for different tiles
    private final char WALL = '*';
    private final char FLOOR = '#';
    private final char ENDPOINT = '^';
    private final char COIN = '%';
    private final char QUESTION = '?';
    private final char CHERRY = '"';
    private final char GEM = '~';

    // constants for the game states
    private final int MAIN_MENU = 1;
    private final int PLAYING_GAME = 2;
    private final int HELP_MENU = 3;
    private final int END_SCREEN = 4;
    private final int QUESTION_SCREEN = 5;
    private final int ANSWER_SCREEN = 6;
    private final int CORRECT_ANSWER = 7;
    private final int WRONG_ANSWER = 8;

    // arraylist for questions
    ArrayList<Question> questions = new ArrayList<Question>();

    // constructor
    public GameFrame(String t, int w, int h){
        super(t, w, h);
        // loading sprites and blocks
        try {
            wall = ImageIO.read(new File(WALL_SPRITE));
            floor = ImageIO.read(new File(FLOOR_SPRITE));
            endPoint = ImageIO.read(new File(ENDPOINT_SPRITE));
            coin = ImageIO.read(new File(COIN_SPRITE));
            question = ImageIO.read(new File(QUESTION_SPRITE));
            cherry = ImageIO.read(new File(CHERRY_SPRITE));
            gem = ImageIO.read(new File(GEM_SPRITE));
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error loading sprites of
blocks and pickups\nError Message: " + e.getMessage(), "Blocks and Pickups
Sprites Loading Error", JOptionPane.ERROR_MESSAGE);
        }

        // loading menus and screens
        try {
            mainMenu = ImageIO.read(new File(MAIN_MENU_SCREEN));
            helpMenu = ImageIO.read(new File(HELP_MENU_SCREEN));
            questionScreen = ImageIO.read(new
File(QUESTION_TEMPLATE_SCREEN));
            correctAnswer = ImageIO.read(new File(CORRECT_SCREEN));
            wrongAnswer = ImageIO.read(new File(WRONG_SCREEN));
            endScreen = ImageIO.read(new File(END_MENU_SCREEN));
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error loading menus\nError
Message: " + e.getMessage(), "Menus Loading Error",
JOptionPane.ERROR_MESSAGE);
        }
        quit = false;

        offScreen = new BufferedImage(640, 480,
BufferedImage.TYPE_INT_ARGB);

        level = new Level();
        level.loadLevel("maze1.txt");

        loadQuestions();

        // starting the animation thread
        animation = new Thread(this);
        animation.start();

        player = new Player(FACING_DOWN, 32, 20, 32, 32);
        player.setVisible(true);

        addKeyListener(this);
    }

    public void drawLevel(){
        int x = 0;
        int y = 25;
        Graphics g = offScreen.getGraphics();

        for(int i = 0; i < 15; i++){ // loop for x
            for(int j = 0; j < 20; j++){ //loop for y
                switch(level.getTile(j, i)){ //using character from get
tile to decide which block
                    case WALL:   g.drawImage(wall, x, y, null);
                    break;
                    case FLOOR:   g.drawImage(floor, x, y, null);
                    break;
                    case ENDPOINT:   g.drawImage(endPoint, x, y, null);
                    break;
                    case COIN:   g.drawImage(coin, x, y, null);
                    break;
                    case QUESTION: g.drawImage(question, x, y, null);
                    break;
                    case CHERRY: g.drawImage(cherry, x, y, null);
                    break;
                    case GEM: g.drawImage(gem, x, y, null);
                    break;
                    default: JOptionPane.showMessageDialog(null, "Error
loading tiles into level", "Tiles Loading Error",
JOptionPane.ERROR_MESSAGE);
                }
                x = x + 32; //to move a block to the right
            }

            y = y + 32; //to move a block down
            x = 0; //to start drawing from the top again
        }
    }

    @Override
    public void run(){
        while(quit == false){
            renderGame();
            repaint();
            try{
                Thread.sleep(30); // Make the tread sleep to give time for
other threads to be processed
            } catch (Exception e){
                JOptionPane.showMessageDialog(null, "Error wtih
Thread\nError Message: " + e.getMessage(), "Thread Error",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    public void renderGame(){
        Graphics g = offScreen.getGraphics();
        if(gameState == MAIN_MENU){
            g.drawImage(mainMenu, 0, 0, null);
        } else if(gameState == PLAYING_GAME){
            drawLevel();
            player.paint(g);

            g.setFont(new Font("Helvetica", Font.BOLD, 28));
            g.setColor(Color.WHITE);

            g.drawString("Score: " + score, 475, 60);
        } else if(gameState == HELP_MENU){
            g.drawImage(helpMenu, 0, 0, null);
        } else if(gameState == END_SCREEN){
            g.drawImage(endScreen, 0, 0, null);

            g.setFont(new Font("Helvetica", Font.BOLD, 37));
            g.setColor(Color.WHITE);

            g.drawString("" + score, 500, 250);
        } else if(gameState == QUESTION_SCREEN){
            g.drawImage(questionScreen, 0, 0, null);

            int ran;

            do {
                ran = (int)(Math.random() * questions.size()); // choosing
random question
            } while(questionsUsed[ran] == true);

            g.setColor(Color.BLACK);
            g.setFont(new Font("Century", Font.BOLD, 18));

            currQuestion = questions.get(ran);
            questionsUsed[ran] = true;

            g.drawString(currQuestion.getQuestion(), 55, 75);

            g.setFont(new Font("Century", Font.BOLD, 20));

            g.drawString(currQuestion.getOp1(), 150, 175);
            g.drawString(currQuestion.getOp2(), 150, 255);
            g.drawString(currQuestion.getOp3(), 150, 335);
            g.drawString(currQuestion.getOp4(), 150, 415);

            gameState = ANSWER_SCREEN;
        } else if(gameState == ANSWER_SCREEN){
            // nothing since its waiting for user to answer
        } else if(gameState == CORRECT_ANSWER){
            g.drawImage(correctAnswer, 0, 0, null);
        } else if(gameState == WRONG_ANSWER){
            g.drawImage(wrongAnswer, 0, 0, null);
        }

    }

    public void loadQuestions(){
        try{
            BufferedReader f = new BufferedReader(new
FileReader("questions.txt"));

            String line; //for the BR to read from

            while((line = f.readLine()) != null){
                Question q = new Question();

                q.setQuestion(line);
                q.setOp1(f.readLine());
                q.setOp2(f.readLine());
                q.setOp3(f.readLine());
                q.setOp4(f.readLine());
                q.setAnswer(Integer.parseInt(f.readLine()));

                questions.add(q); //put all questions with variables in an
arraylist
            }
            f.close();

            //created boolean array to avoid duplicate questions
            questionsUsed = new boolean [questions.size()];
            for(boolean question : questionsUsed){
                question = false;
            }

            //System.out.println(questions.size() + " questions loaded");
        } catch (Exception e){
            JOptionPane.showMessageDialog(null, "Error loading
questions\nError Message: " + e.getMessage(), "Questions Loading Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    @Override
    public void update(Graphics g){
        paint(g);
    }

    @Override
    public void paint(Graphics g){
        if(offScreen != null){
            g.drawImage(offScreen, 1, 1, null);
            g.setColor(Color.BLACK);
            g.fillRect(0, 480, 640, 10);
        }
    }

    public void newGame(){
        score = 0;
        level.loadLevel("maze1.txt");
        player.setLocation(32, 20);
    }

    public void changeLevel(){
        if(levelNo == 1){
            level.loadLevel("maze2.txt");
            levelNo++;
            player.setLocation(32, 20);
        } else if(levelNo == 2){
            level.loadLevel("maze3.txt");
            levelNo++;
            player.setLocation(32, 20);
        } else if(levelNo == 3){
            gameState = END_SCREEN;
        }
    }

    // checks the blocks adjacent to character
    public char tileToRight(int x, int y){
        int posX = (x/32) + 1;
        int posY = (y/32);
        if(posX < 20){
            return level.getTile(posX, posY);
        } else {
            return FLOOR;
        }
    }

    public char tileToLeft(int x, int y){
        int posX = (x/32) - 1;
        int posY = (y/32);
        if(posX < 20){
            return level.getTile(posX, posY);
        } else {
            return FLOOR;
        }
    }

    public char tileToDown(int x, int y){
        int posX = (x/32);
        int posY = (y/32) + 1;
        if(posY < 15){
            return level.getTile(posX, posY);
        } else {
            return FLOOR;
        }
    }

    public char tileToUp(int x, int y){
        int posX = (x/32);
        int posY = (y/32) - 1;
        if(posY < 15){
            return level.getTile(posX, posY);
        } else {
            return FLOOR;
        }
    }

    public void checkNextTile(){
        int px = player.getX();
        int py = player.getY();
        if(level.getTile(px/32, py/32) == COIN || level.getTile(px/32,
py/32) == CHERRY || level.getTile(px/32, py/32) == GEM){
            level.setTile(px/32, py/32, FLOOR);
            score++;
        } else if (level.getTile(px/32, py/32) == ENDPOINT){
            changeLevel();
        } else if (level.getTile(px/32, py/32) == QUESTION){
            gameState = QUESTION_SCREEN;
            level.setTile(px/32, py/32, FLOOR);
        }
    }

    @Override
    public void keyPressed(KeyEvent e){
        int px = player.getX();
        int py = player.getY();

        if(e.getKeyCode() == KeyEvent.VK_LEFT || e.getKeyCode() ==
KeyEvent.VK_A){
            if(tileToLeft(px, py) != WALL){
                player.moveLeft();

                px = player.getX(); //getting updated x and y postitions
                py = player.getY();

                checkNextTile();
            }
        }

        if(e.getKeyCode() == KeyEvent.VK_RIGHT || e.getKeyCode() ==
KeyEvent.VK_D){
            if(tileToRight(px, py) != WALL){
                player.moveRight();

                px = player.getX();
                py = player.getY();

                checkNextTile();
            }
        }

        if(e.getKeyCode() == KeyEvent.VK_DOWN || e.getKeyCode() ==
KeyEvent.VK_S){
            if(tileToDown(px, py) != WALL){
                player.moveDown();

                px = player.getX();
                py = player.getY();

                checkNextTile();
            }
        }

        if(e.getKeyCode() == KeyEvent.VK_UP || e.getKeyCode() ==
KeyEvent.VK_W){
            if(tileToUp(px, py) != WALL){
                player.moveUp();

                px = player.getX();
                py = player.getY();
                checkNextTile();
            }
        }

        if(e.getKeyCode() == KeyEvent.VK_F1){
            if(gameState == MAIN_MENU){
                newGame();
                gameState = PLAYING_GAME;
            } else if(gameState == HELP_MENU){
                gameState = MAIN_MENU;
            } else if(gameState == END_SCREEN){
                gameState = MAIN_MENU;
            }
        }

        if(e.getKeyCode() == KeyEvent.VK_F2){
            if(gameState == MAIN_MENU){
                gameState = HELP_MENU;
            }
        }

        if(e.getKeyCode() == KeyEvent.VK_F3){
            if(gameState == MAIN_MENU){
                quit = true;
                System.exit(0);
            }
        }

        if(e.getKeyCode() == KeyEvent.VK_1){
            if(gameState == ANSWER_SCREEN){
                if(1 == currQuestion.getAnswer()){
                    score+=2;
                    gameState = CORRECT_ANSWER;
                } else {
                    gameState = WRONG_ANSWER;
                }
            }
        }

        if(e.getKeyCode() == KeyEvent.VK_2){
            if(gameState == ANSWER_SCREEN){
                if(2 == currQuestion.getAnswer()){
                    score+=2;
                    gameState = CORRECT_ANSWER;
                } else {
                    gameState = WRONG_ANSWER;
                }
            }
        }

        if(e.getKeyCode() == KeyEvent.VK_3){
            if(gameState == ANSWER_SCREEN){
                if(3 == currQuestion.getAnswer()){
                    score+=2;
                    gameState = CORRECT_ANSWER;
                } else {
                    gameState = WRONG_ANSWER;
                }
            }
        }
```

```
        if(e.getKeyCode() == KeyEvent.VK_4){
            if(gameState == ANSWER_SCREEN){
                if(4 == currQuestion.getAnswer()){
                    score+=2;
                    gameState = CORRECT_ANSWER;
                } else {
                    gameState = WRONG_ANSWER;
                }
            }
        }

        if(e.getKeyCode() == KeyEvent.VK_SPACE){
            if(gameState == CORRECT_ANSWER || gameState == WRONG_ANSWER){
                gameState = PLAYING_GAME;
            }
        }
    }

    @Override
    public void keyReleased(KeyEvent e){

    }

    @Override
    public void keyTyped(KeyEvent e){

    }

    @Override
    public void onClosing(){
        quit = true;
        System.exit(0);
    }
}
```

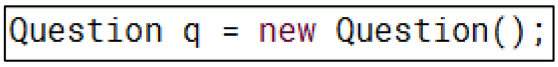
Main:

```
public class Main {
    public static void main (String args[]){
        GameFrame f = new GameFrame("Maze of Wit", 640, 480);
        f.setVisible(true);
        f.setResizable(false);
        f.setLocationRelativeTo(null); // centre the frame when it comes up
    }
}
```

3.2 – Notes on the Printout

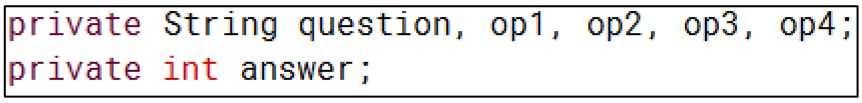
Object Oriented Programming Principles

OOP Principles can be seen throughout the whole program with principles such as the use of objects and encapsulation. In the below screenshot it can be seen that objects are being used to create an object of type ‘Question’. This is useful as it is much more efficient as there is no need to make structures and methods that are similar or the same in each class.

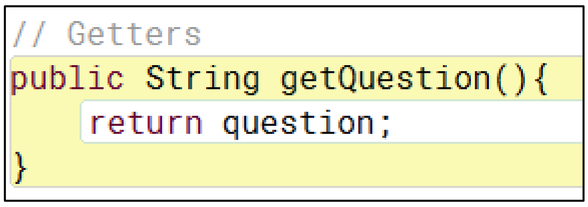


```
Question q = new Question();
```

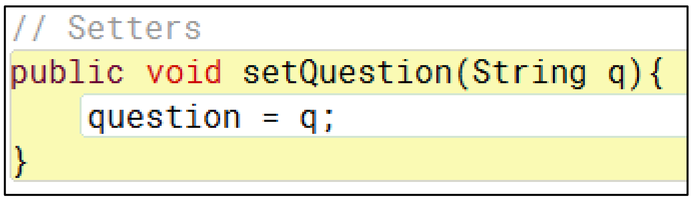
Another advantage of objects is that with them it is possible to use encapsulation. As seen in the below photos the variables of the class question are all private. However, they still need to be accessed by other classes. To accomplish this, methods known as ‘getters’ and ‘setters’ are made in order to access the data of the variables. Encapsulation is important as it keeps the integrity of the program intact and it will not allow directly tampering with the variables.



```
private String question, op1, op2, op3, op4;
private int answer;
```



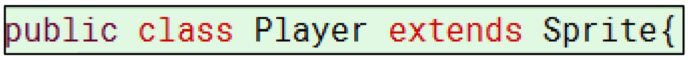
```
// Getters
public String getQuestion(){
    return question;
}
```



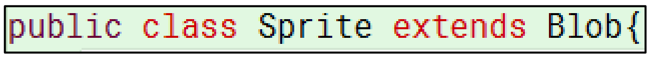
```
// Setters
public void setQuestion(String q){
    question = q;
}
```

Inheritance

Inheritance is used in this program in order to increase the amount of reusable code throughout the application and thus decrease chances of errors and overall efficiency. Programs will take the variables and methods of the class that they are extended from. However, if a method of a sub-class needs to be changed this can be done easily by overriding it.



```
public class Player extends Sprite{
```

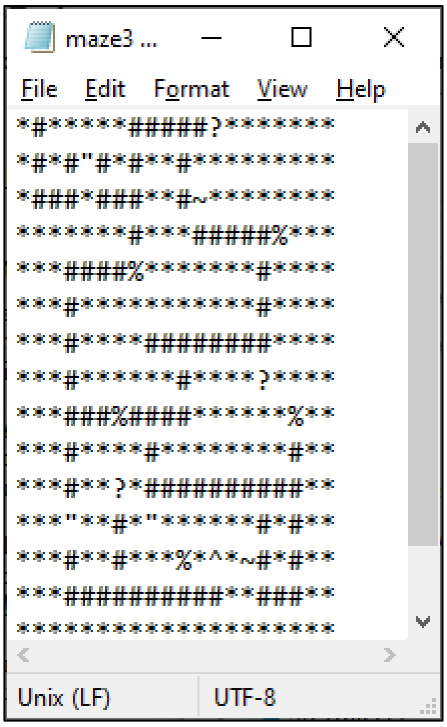


```
public class Sprite extends Blob{
```

In this case the class ‘Player’ is being extended from the class ‘Sprite’ which in turn itself is being extended from the class ‘Blob’. Therefore, the classes that are used in ‘Blob’ and ‘Sprite’ can be and are also used in the player class.

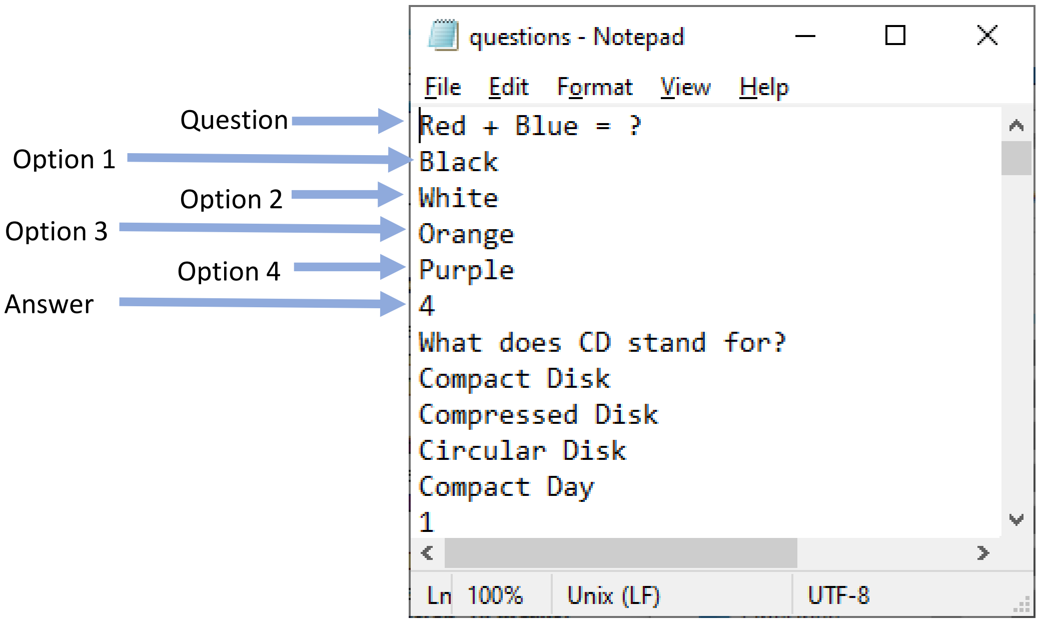
Use of Files

Text files were used thoroughly in this application in order to keep things simple and to make it much easier for future improvements and additions. Firstly, three text files were used in order to keep the map of each maze on each level. The mazes are saved as characters in which a character would correspond to a block on screen when the game is rendered. This way it is much easier to change levels as all that needs to be done is change the characters in the text file. An example of this can be seen below.



```
*#*****#####?*******
*#*#"#*#**#*********
*###*###**#~********
*******#***#####%***
***####%*******#****
***#***********#****
***#****########****
***#******#****?****
***###%####******%**
***#****#********#**
***#**?*##########**
***"**#*"******#*#**
***#**#***%*^*~#*#**
***##########**###**
********************
```

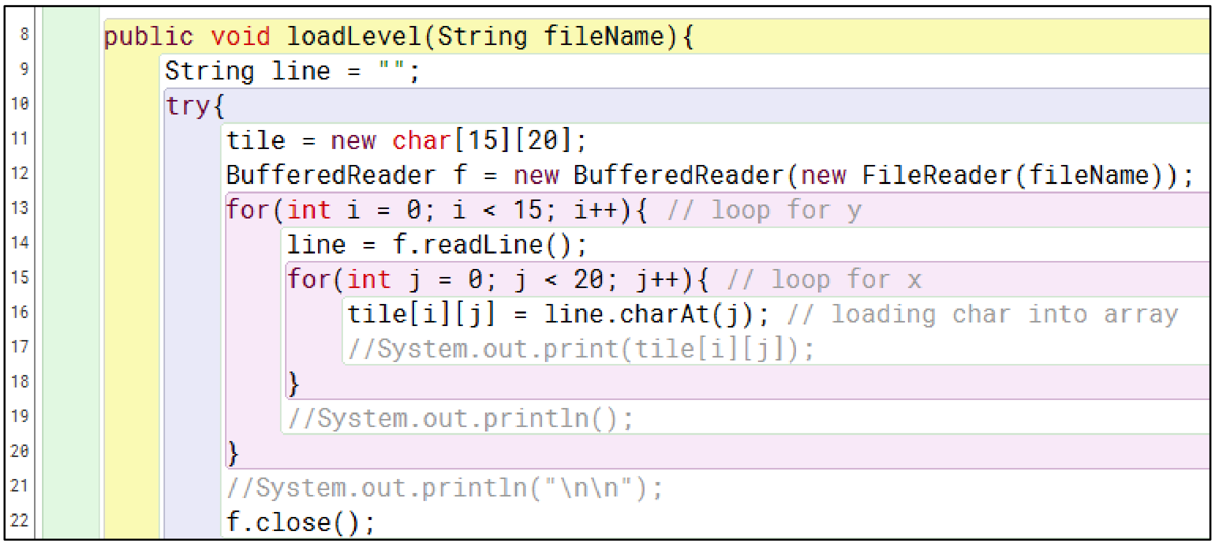
In addition, another text tile was used to store the questions that will be asked in the game. The way that they were kept was that first there was the actual question, then with all of their own lines, the four options and followed by a numerical value between one and four to signal which of the options is correct. It is more efficient this way as if Mr. Sciberras wants to add more questions he can do so very easily by just typing them at the end of the file. It is also not difficult to change or edit a question.



```
Red + Blue = ?
Black
White
Orange
Purple
4
What does CD stand for?
Compact Disk
Compressed Disk
Circular Disk
Compact Day
1
```

File Operations

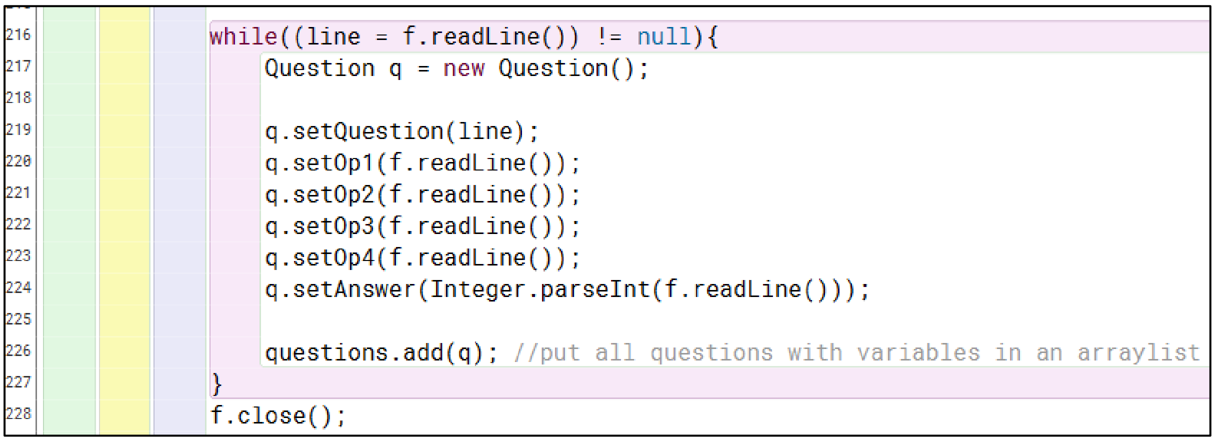
The above files obviously had to be read in order to be processed and used in the application in both of their respective uses.



```
8   public void loadLevel(String fileName){
9       String line = "";
10      try{
11          tile = new char[15][20];
12          BufferedReader f = new BufferedReader(new FileReader(fileName));
13          for(int i = 0; i < 15; i++){ // loop for y
14              line = f.readLine();
15              for(int j = 0; j < 20; j++){ // loop for x
16                  tile[i][j] = line.charAt(j); // loading char into array
17                  //System.out.print(tile[i][j]);
18              }
19              //System.out.println();
20          }
21          //System.out.println("\n\n");
22          f.close();
```

In the above screenshot we can see the file being read from in the fourteenth line. In this case it is reading each character in the line and when it is done it will then move on to the next line with the help of nested loops.

In the below screenshot it can be seen that the files are being read from in order to get the attributes of a question from the file into an object of type ‘Question’.



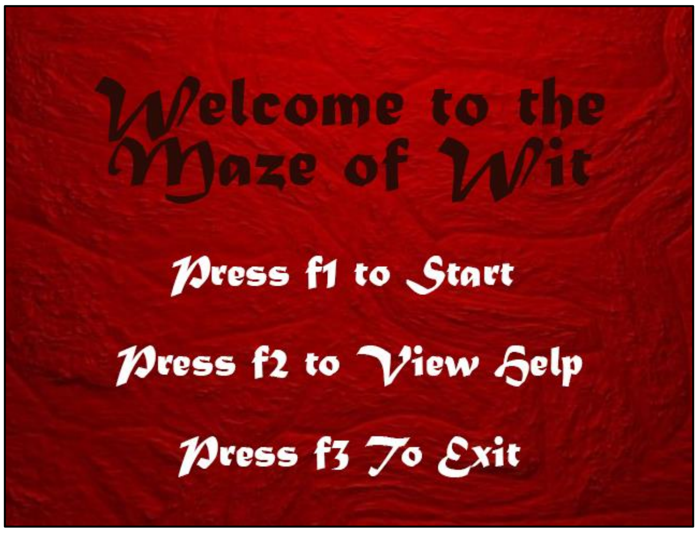
```
while((line = f.readLine()) != null){
    Question q = new Question();

    q.setQuestion(line);
    q.setOp1(f.readLine());
    q.setOp2(f.readLine());
    q.setOp3(f.readLine());
    q.setOp4(f.readLine());
    q.setAnswer(Integer.parseInt(f.readLine()));

    questions.add(q); //put all questions with variables in an arraylist
}
f.close();
```

Applications of Java API

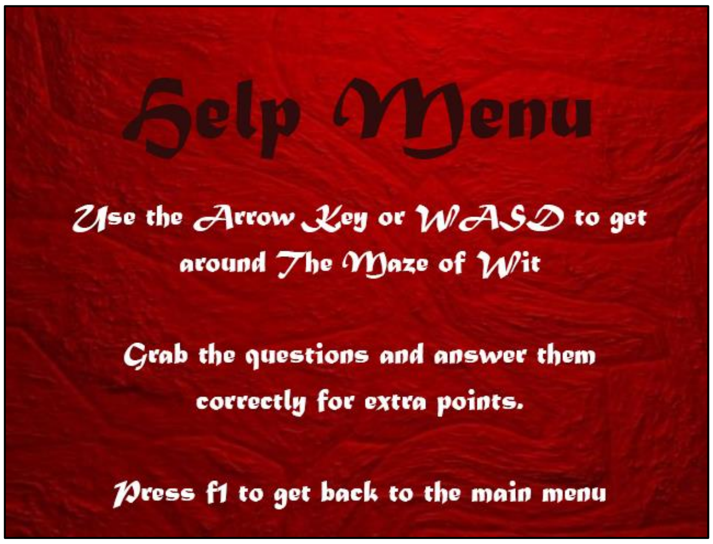
Different packages from the Java API were used in the program more often than not. Some of these packages that were used are the AWT, Frames, BufferedImages, JOptionPane, KeyListener and BufferedReader. The AWT was used for various reasons such as the rectangle for the area and more importantly the graphics object that was used a lot throughout the making of the application. The frame was primarily used for the 'ClosingFrame' class that allowed the frame to be closed from the top right 'x' button. It was also used in order to actually make the screens appear on the screen. BufferedImages are extremely important as it allowed for sprites and blocks to have pictures and images. This made the game much more attractive and eye-catching as opposed to having everything text based. The JOptionPane was used for error catching mostly. Whenever there was a try-catch loop, the catch part would have JOptionPane to display the error. This made it look more professional as opposed to having it simply printed on the screen. The KeyListener was obviously used to allow to user to use the keys on the keyboard to move the character and advance through the game. Finally, the BufferedReader was used in order to read from text files for the questions and the mazes as previously explained.

Interface Efficiency

The interface was extremely simple to use as when the program was opened the user is then given three options, to either press 'F1' to start the game, press 'F2' to view the help and 'F3' to exit the game. This can be seen below.



From the help menu you can then press ‘F1’ to go back to the main menu.



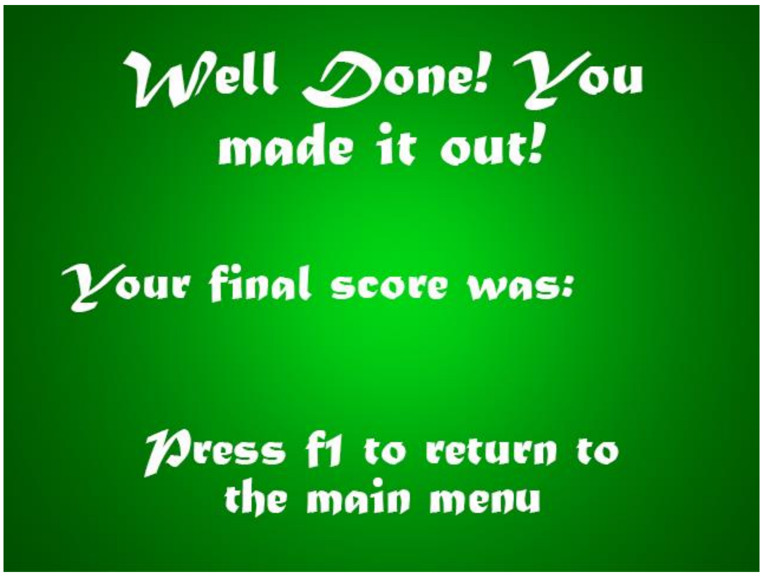
Help Menu

Use the Arrow Key or WASD to get around The Maze of Wit

Grab the questions and answer them correctly for extra points.

Press f1 to get back to the main menu

Finally, when the game is finished, the player is then greeted with their final score and an option to press ‘F1’ to get back to the main menu.



Well Done! You made it out!

Your final score was:

Press f1 to return to the main menu

This way of navigation is extremely simple and smooth. This way it is easy for a user who has never used the program before to know how to get around it. Meaning the interface is as user-friendly as can be.

Program Testing

4.1 - Testing Description

The tests will be split into three sections; File Handling, Character Movement and Question Testing.

File Handling:

1. The first test that will be done is how the program will behave when a file is missing. This test will involve the removal of a file and then calling for it in the program to see what will be the outcome.
2. The second test is what will happen if a file in the program does not have the correct file extension that it should have.
3. This test will include what will happen if a file has an incorrect name that it should not have.
4. The test that will be done now is how the application will behave when there is a line missing from a text file.
5. Similar to the previous test this one will have an extra line in a file that the program is reading from.

Character Movement:

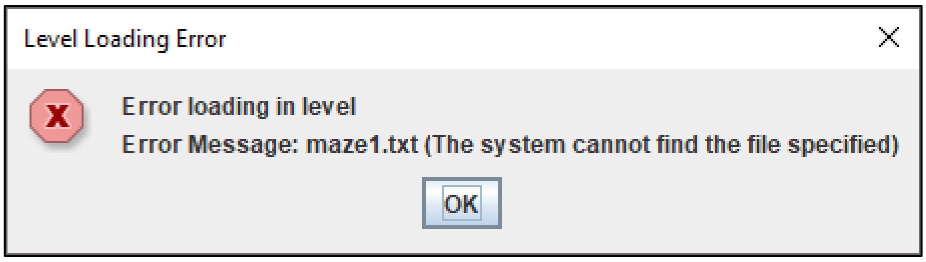
6. Checking if the WASD keys as well as the arrow keys are both good to use is what will be done in this test.
7. Regarding this test it will be seen what will happen when the user decides to move into a wall.
8. What will be the outcome when the user will want to move in an empty space or route.
9. In this test the chance of a wrong input to move will be checked to see how the application will react.

Question Testing:

10. This test will include checking whether the same question can come out multiple times during one playthrough.
11. Another test that will be done is to see what will happen if the user presses a wrong input that is not expected by the program.
12. Checking what will happen when the user responds to a question with the correct answer.
13. Finally, it will be seen how the application will respond when the user gets a question wrong.

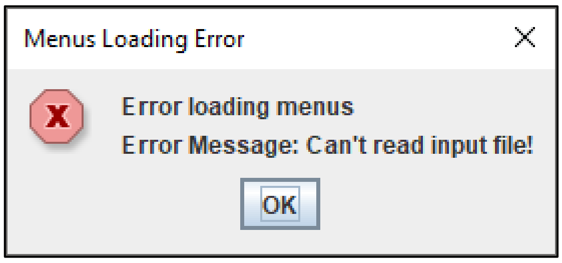
4.1 – Evidence of Testing

Test Case No.	Input	Expected Output	Actual Output
1	File Removed	Error Message	Error Message
2	File has incorrect file extension	Error Message	Error Message
3	File has incorrect name	Error Message	Error Message
4	Text file has less amount of lines	Error Message	Error Message
5	Text file has more amount of lines	Extra Lines will be ignored	Extra Lines were ignored
6	WASD and Arrow keys are used to move	Character will move	Character moved
7	Character moving into wall	Character will not move	Character did not move
8	Character moving into empty space	Character will move	Character moved
9	Using the wrong keys to move	Input ignored	Input ignored
10	Playing the whole game	No repeated questions	No repeated questions
11	Pressing the wrong button when responding a question	Input ignored	Input ignored
12	Getting a question correct	Screen showing answer is correct will show	Screen showing answer is correct was shown
13	Getting a question wrong	Screen showing answer is wrong will show	Screen showing answer is wrong was shown


Level Loading Error

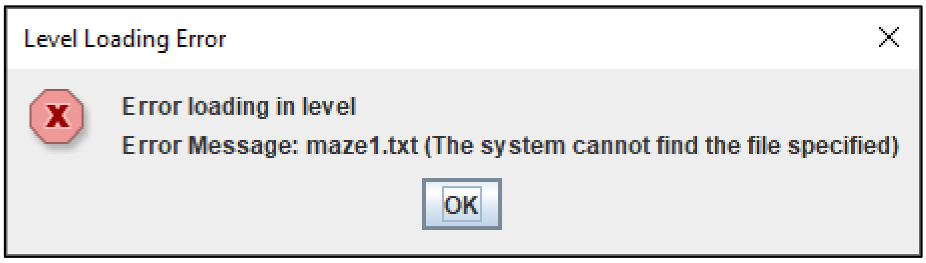
Error loading in level
Error Message: maze1.txt (The system cannot find the file specified)

Screenshot for Test 1


Menus Loading Error

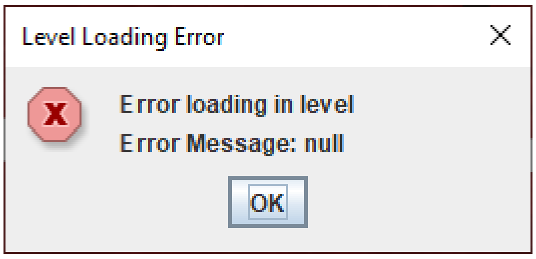
Error loading menus
Error Message: Can't read input file!

Screenshot for Test 2


Level Loading Error

Error loading in level
Error Message: maze1.txt (The system cannot find the file specified)

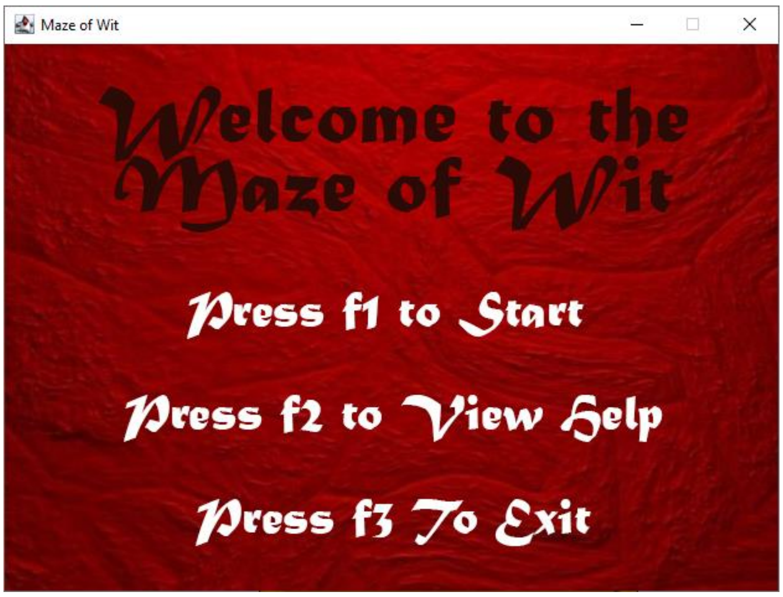
Screenshot for Test 3



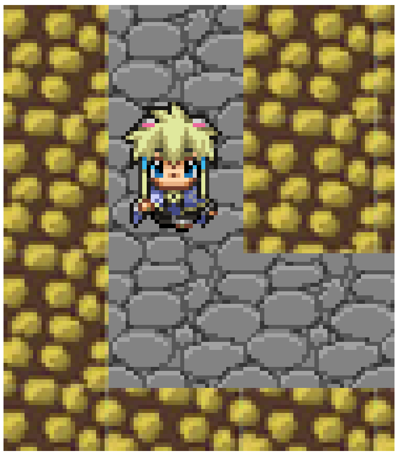
Level Loading Error

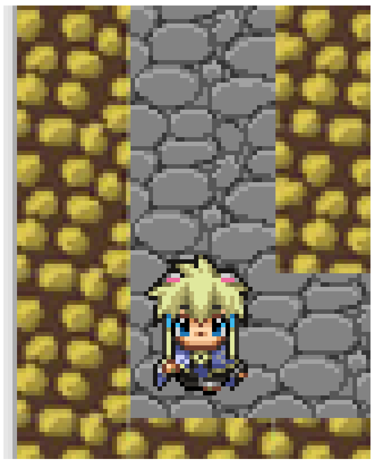
Error loading in level
Error Message: null

Screenshot for Test 4

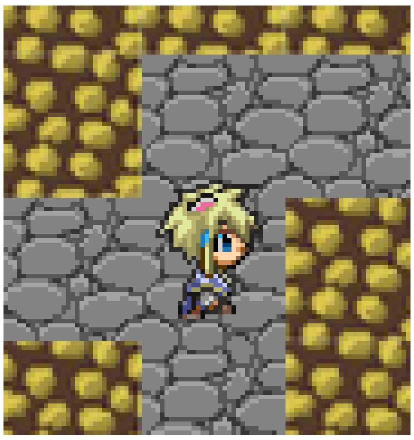


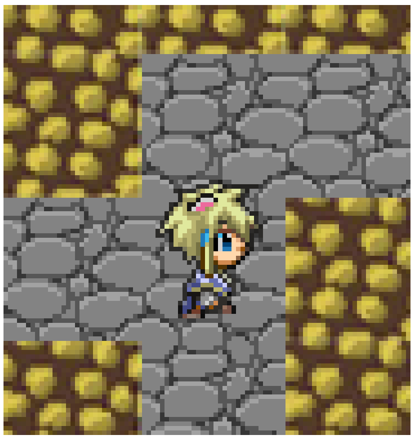
Screenshot for Test 5


Before S or Down arrow key is pressed

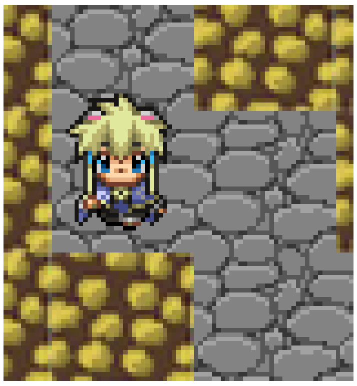

After S or Down arrow key is pressed

Screenshots for Test 6

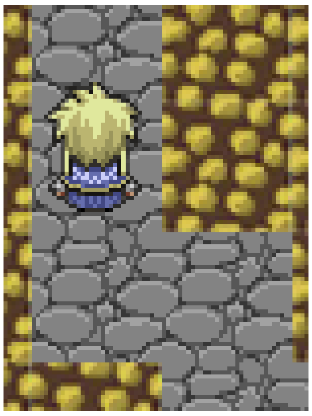

Before D or right arrow key is pressed


After D or right arrow key is pressed

Screenshots for Test 7

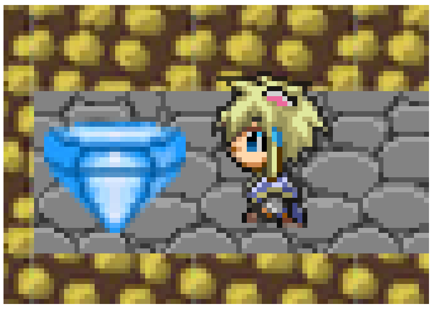


Before W or up arrow key is pressed

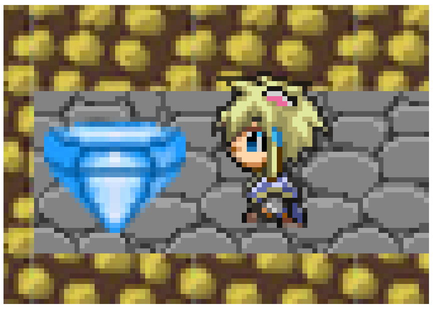


After W or up arrow key is pressed

Screenshots for Test 8

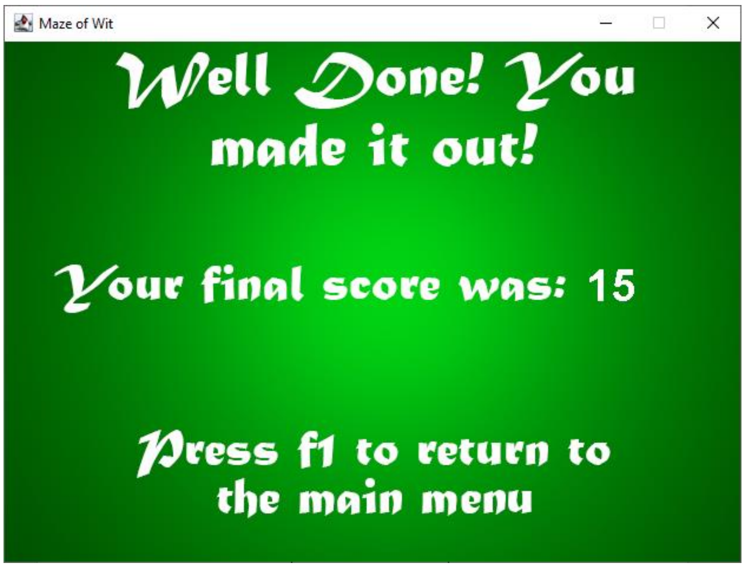


Before G key is pressed



After G key is pressed

Screenshots for Test 9

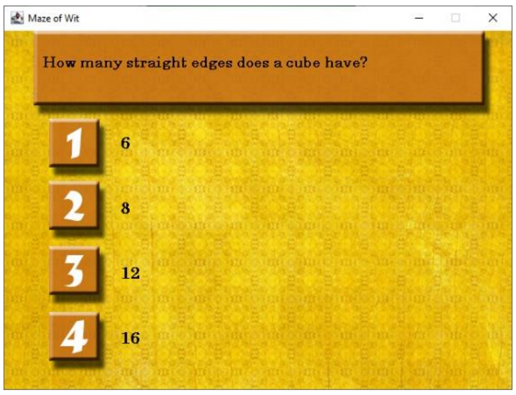


Well Done! You made it out!

Your final score was: 15

Press f1 to return to the main menu

Screenshot for Test 10



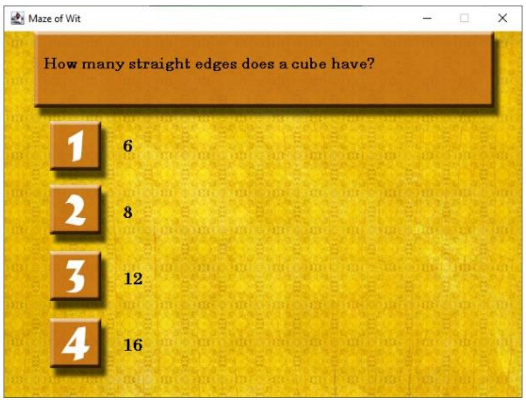
How many straight edges does a cube have?

6

8

12

16



How many straight edges does a cube have?

6

8

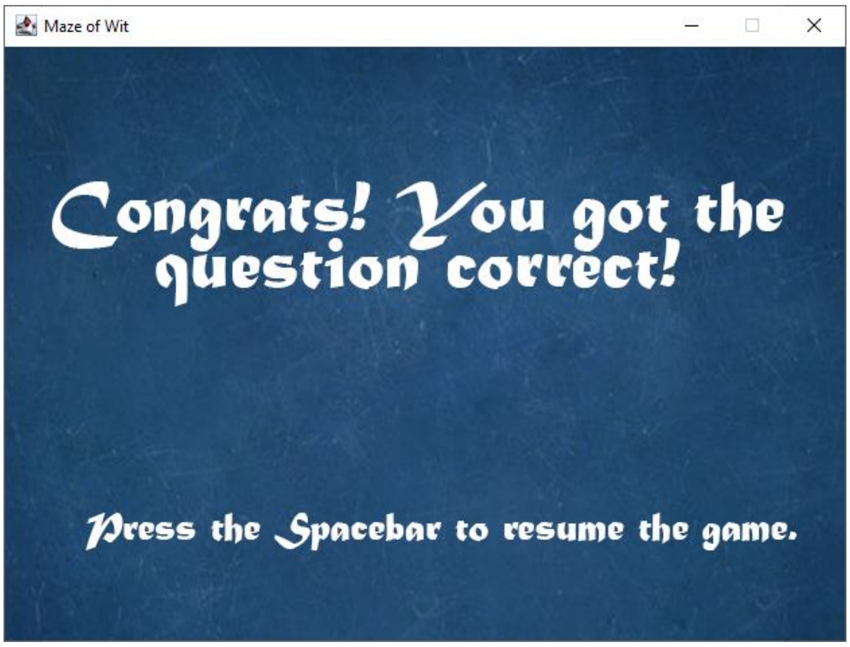
12

16

Before the 6 button was pressed

After the 6 button was pressed

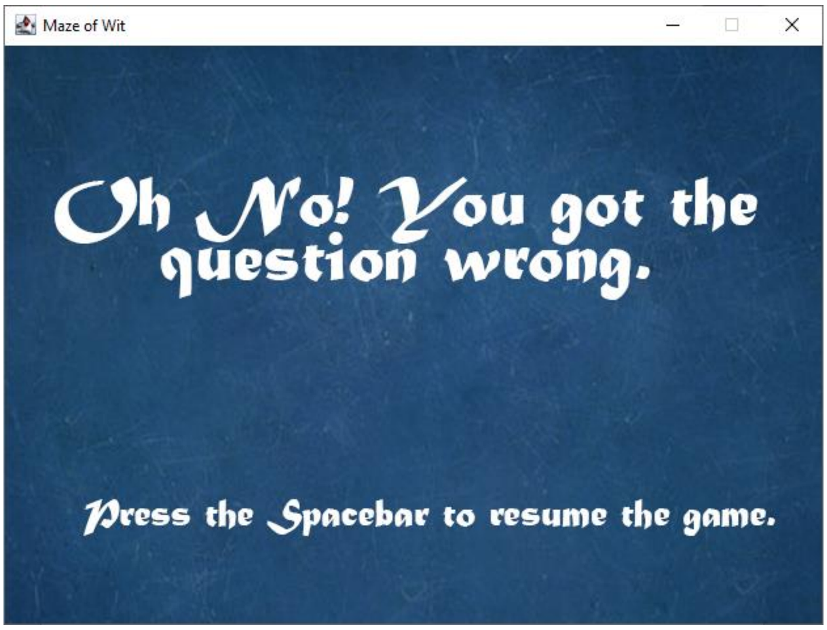
Screenshot for Test 11


Maze of Wit

Congrats! You got the question correct!

Press the Spacebar to resume the game.

Screenshot for Test 12

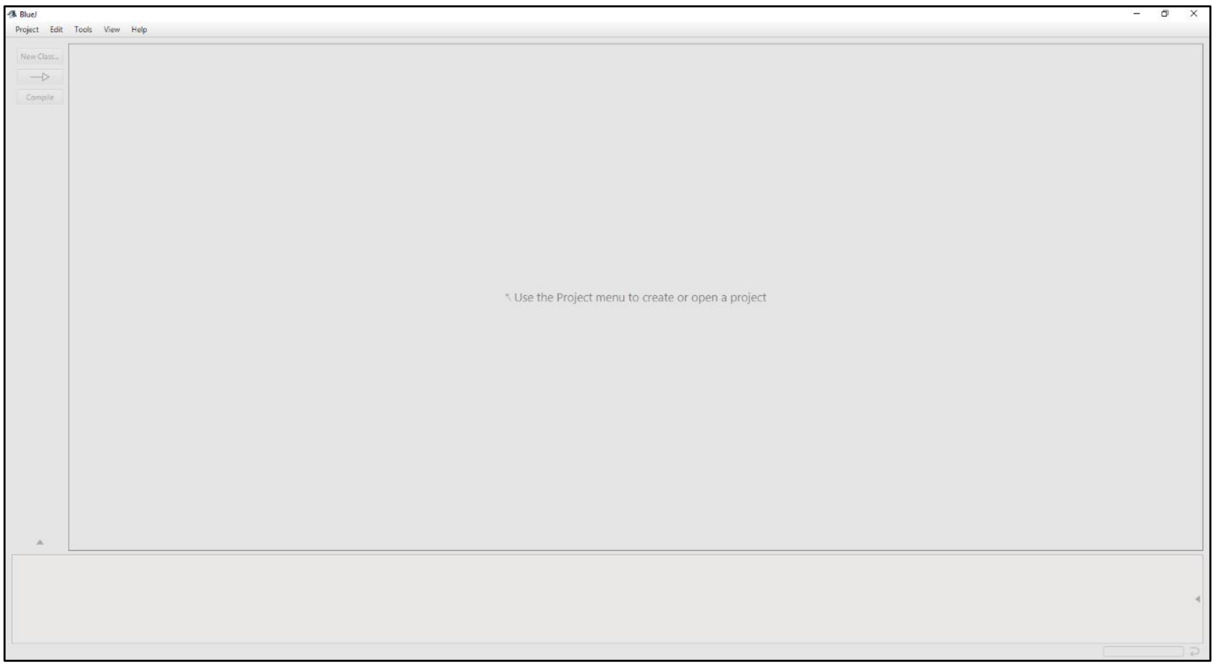

Maze of Wit

Oh No! You got the question wrong.

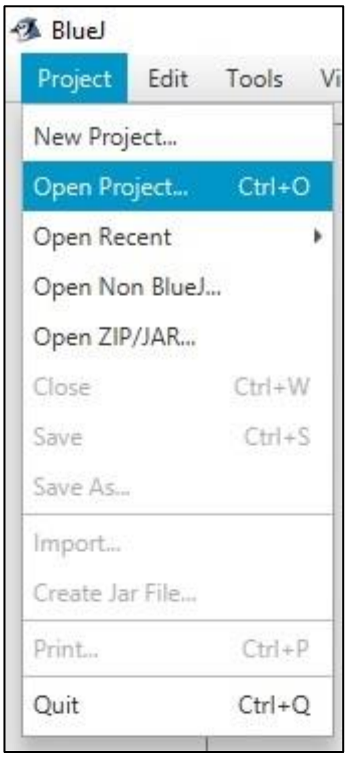
Press the Spacebar to resume the game.

Screenshot for Test 13

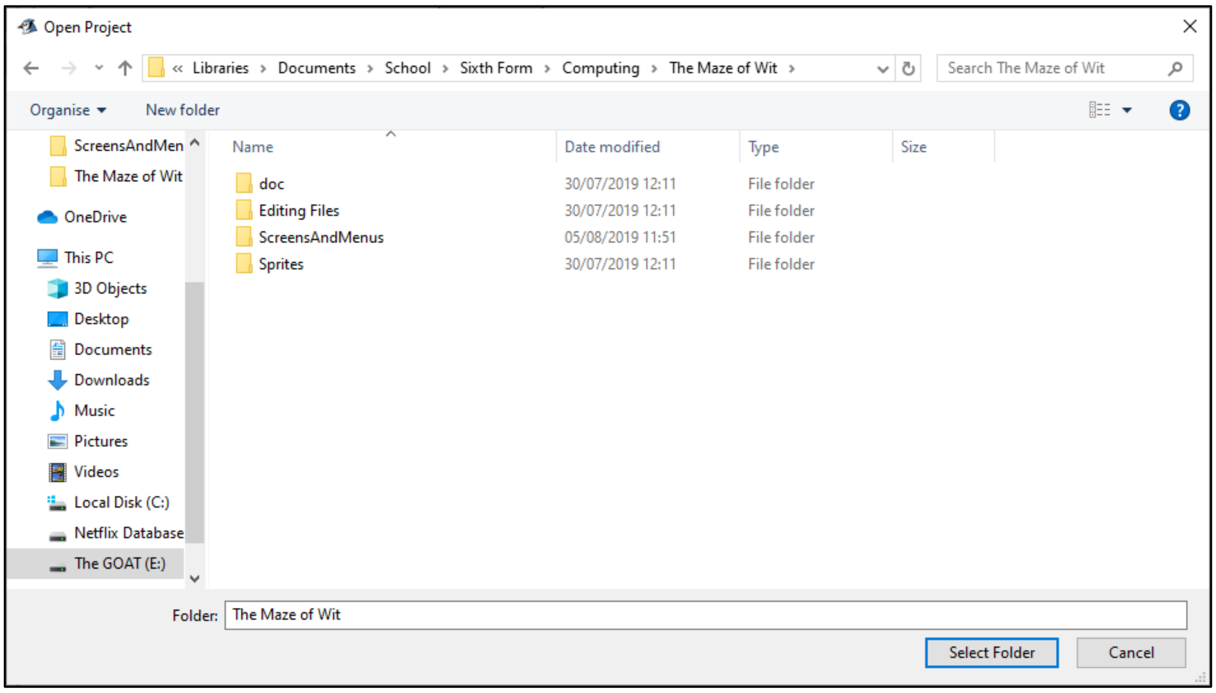
User's Manual


BlueJ
Project Edit Tools View Help
New Class...
Compile
Use the Project menu to create or open a project

1. Open BlueJ



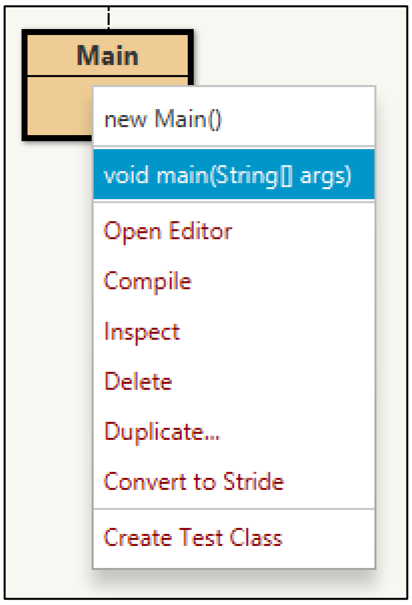
2. Press the ‘Project’ tab and click on ‘Open Project’.



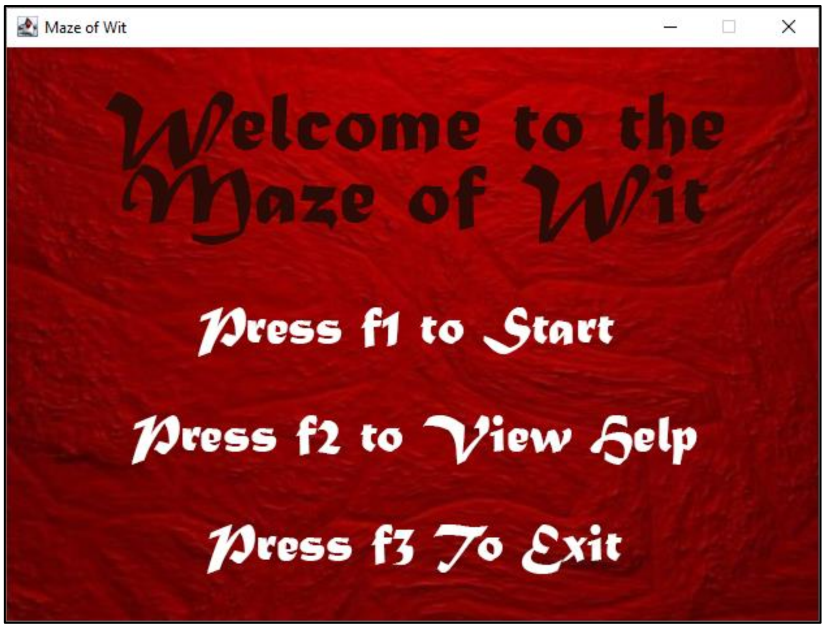
Name	Date modified	Type	Size
doc	30/07/2019 12:11	File folder	
Editing Files	30/07/2019 12:11	File folder	
ScreensAndMenus	05/08/2019 11:51	File folder	
Sprites	30/07/2019 12:11	File folder	

Folder: The Maze of Wit

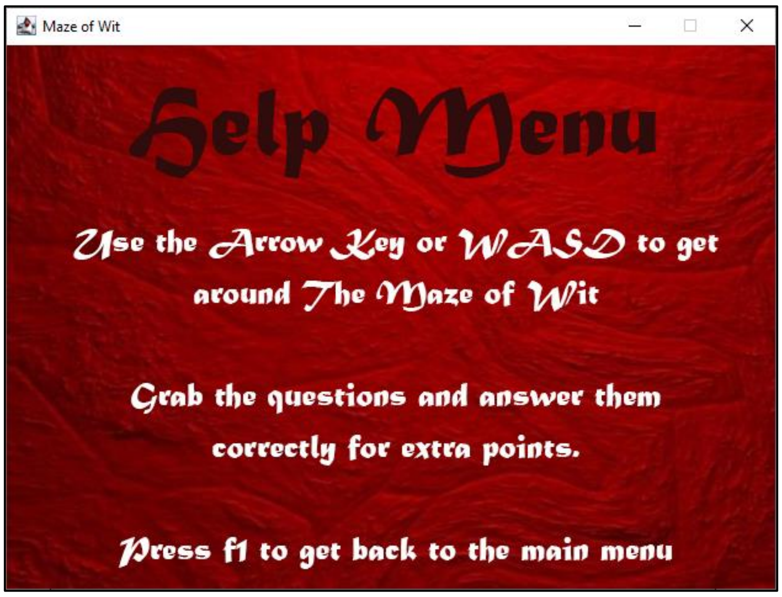
3. Find the file that the programs are saved in and press ‘Select Folder’.



4. Right click on the ‘Main’ class and select the ‘void main(String[] args)’ option.



5. Follow the instructions on screen in order to play the game.



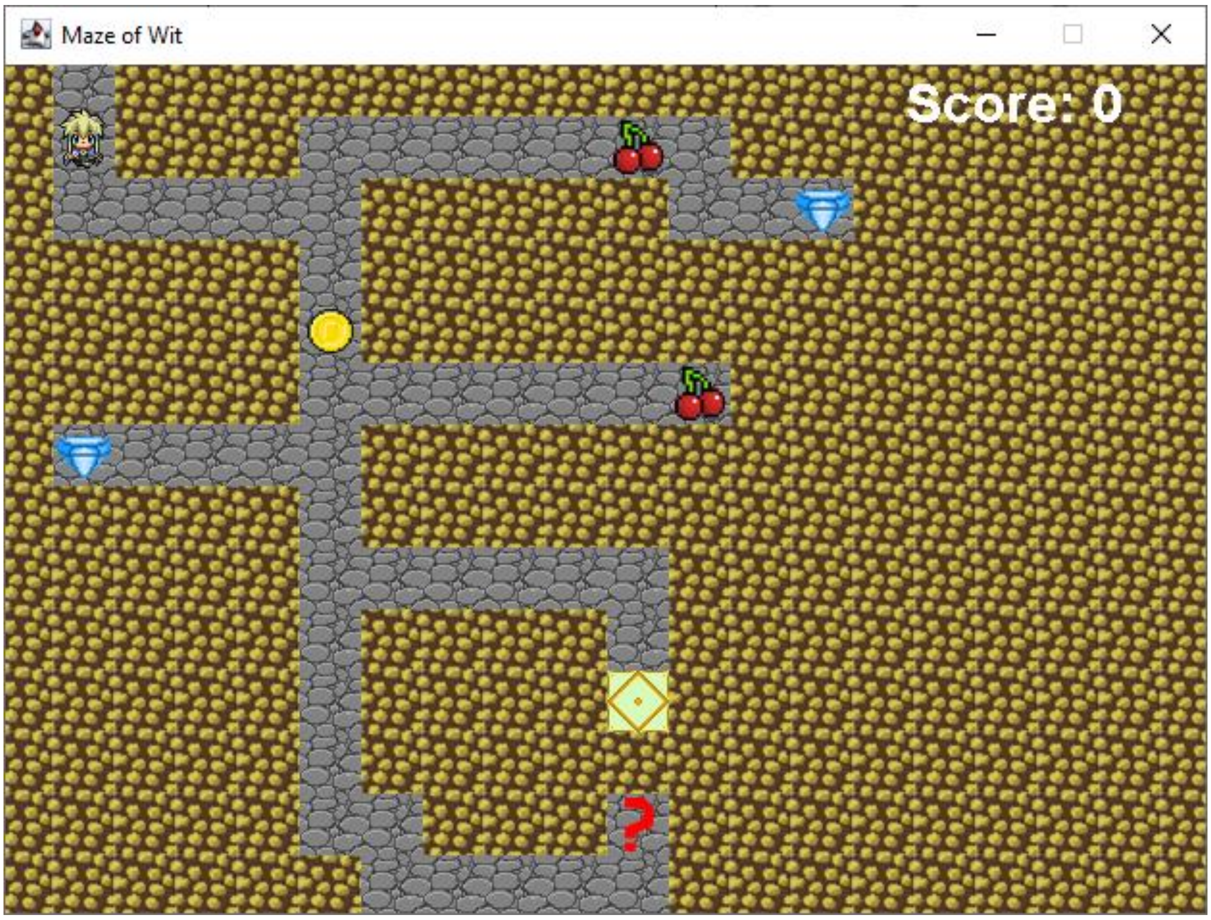
Use the Arrow Key or WASD to get around The Maze of Wit

Grab the questions and answer them correctly for extra points.

Press f1 to get back to the main menu

6. Pressing F2 will take you to the Help Menu where there is more information on how to play the game.

6. Press F1 to return to the Main Menu



7. When the game is started, collect all the gems, coins and cherries to gain as many points as possible. Moving to a question will give you the opportunity to gain more points by answering the questions correctly.

Evaluation and Improvement

6.1 – Evaluation

The final application that was made is an education video game made for children who want to increase the amount of general knowledge they have. This was accomplished with a maze game that has questions that one needs to answer in order to get more points. Apart from this, it is also very easy to add questions and the way that the program was developed makes it very easy to edit and change levels. In addition, the game includes sprites and pictures in order for it to be more captivating and interesting. Moreover, it is not graphically intensive or processor intensive meaning that it can be played on a basic computer without having it crash. Due to the way that the questions that are asked are randomised every time a user plays it also increases the amount of replayability that the game has for the children. Thus, the more they play the more they will learn.

6.2 – Improvement

Even though the application meets all of the requirements that Mr. Sciberras has put out for it, it is not perfect by any means. One of the possible improvements that could have been implemented was another page for the high scores that were achieved. All the scored would have been stored and then put after each other in order of the highest score first and the rest below. Next to each score there would also be the name of the person that got said score.

Another improvement that could have been done is the addition of more levels. Adding more levels with the mazes getting more complex every time is a good way to keep the longevity of the game running. As if there are more levels it will make the users want to play more until they finish and therefore learn more by answering questions along the way.

Moreover, the addition of multiplayer could be extremely fruitful for this application as if two or more users could race to the end of the maze whilst still trying to get as many questions correct as they can, it can be much more exciting and more intense for the users as they have to move as fast as they can.

In addition, adding randomly generated mazes would be a fantastic addition as it will keep the game fresh every time you start a new game. This means that every time you start a new game the maze will be a completely new and playable maze with the pick-ups and available questions scattered along the maze. This will prevent the game from being stale as you will never know what you get when you start a new game. Recursive methods would have been used in order to accomplish this.

Also, animation is something that could have improved the visual quality of the game. Adding animation would have made the game look a lot better and a lot more natural as opposed to having the player jump from one tile to the other. A transition screen from each of the levels when going to a new level could have also be added to make the visual aspect better.

As easy as it is for the teacher to change the levels with the text file, it would be more convenient to have a level editor which is more visual as opposed to just text and character based.

Finally, it would be a possibility that after ever game, the student would write his name and along with their score it would send the teacher a detailed view of what the child got wrong and save it to a text file for the teacher to see.