**הנדסת תוכנה ואבטחת איכות תכנה במערכות מידע – מטלה 4**

Software Development Plan for “Hamka”



**החוג למערכות מידע אוניברסיטת חיפה**

**מגישים:**

|  |  |
| --- | --- |
| **מוחמד אדריס** | 314981598 |
| **אופיר סגל** | **205797210** |
| **שי בן חיים** | 308076553 |
| **ויאם שאהין** | 311182281 |

1. **בדיקות קופסה לבנה:**

|  |  |  |  |
| --- | --- | --- | --- |
| Output | Input | Description | Test ID |
| List of questions | JSON file | Check data imported correctly from JSON file | 1 |
| Updated JSON file | List of questions | Check if data written into JSON file and updated as the user set | 2 |
| Loading game from history | The game the user wants to load | Check if game loaded to user | 3 |
| Game screen with updated board and pawns | Click on “Play” button | Check if board is uploaded with all pawns set | 4 |
| Game screen updated with pawn lost | Pawn move from user in game screen | Check if user can eat opponent pawn | 5 |

1. **בדיקות קופסה שחורה:**

|  |  |  |  |
| --- | --- | --- | --- |
| Output | Input | Description | Test ID |
| Game screen open to user | Button click from user | Check if game started by click on “Play” button | 1 |
| Screen with the question was shown to user | Pawn move on special cell on game screen | Check if question from special cell was shown correctly | 2 |
| GUI timer set to zero | None | Check if timer set to 00:00 | 3 |
| Updated score to player | Pawn move on game screen | Check if player score is updated after he play his turn and set the time | 4 |
| Game screen with user nickname shown | User set his nickname and press button to start a game | check if user set nickname and get shown his name when his turn in game screen | 5 |

1. **קוד JUNIT:**

package Tests;

import static org.junit.Assert.\*;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Scanner;

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import org.junit.Test;

import Utils.PawnColor;

import model.Board;

import model.Cell;

import model.Pawn;

import model.Question;

public class Testing {

/\*\*

\* This method test if the JSON file exist and get imported correctly to the game

\*

\*/

*@Test*

public void testImportJSONFile() {

ArrayList<Question> questions = new ArrayList<Question>();

Object obj = null;

try {

obj = new JSONParser().parse(new FileReader("Template.json"));

} catch (FileNotFoundException e) {

*assertFalse*(false);

} catch (IOException e) {

*assertFalse*(false);

} catch (org.json.simple.parser.ParseException e) {

*assertFalse*(false);

}

try {

JSONObject jo = (JSONObject) obj;

JSONArray arr = (JSONArray) jo.get("questions");

for (Object o : arr) {

JSONObject question = (JSONObject) o;

String content = (String) question.get("question");

JSONArray qs = (JSONArray) question.get("answers");

*@SuppressWarnings*("unchecked")

ArrayList<String> answers = (ArrayList<String>) qs;

String correct = (String) question.get("correct\_ans");

String level = (String)question.get("level");

String team = (String) question.get("team");

Question q = new Question(content, level, answers, correct, team);

*assertNotNull*("cannot add question",questions.add(q));

}

} catch (Exception e) {

*assertFalse*(false);

}

}

/\*\*

\* This method test the move the user want to make in his turn is ok

\*/

*@Test*

public void testGetMove() {

Scanner scan = new Scanner(System.***in***);

int[] toReturn= {0,0,0,0};

System.***out***.println("what pawn to move?");

System.***out***.println("Row:");

String tore0 = scan.nextLine();

toReturn[0] = Integer.*parseInt*(tore0);

System.***out***.println("Column:");

String tore1 = scan.nextLine();

toReturn[1] = Integer.*parseInt*(tore1);

System.***out***.println("move "+toReturn[0]+","+toReturn[1]+" to?");

System.***out***.println("Row:");

String tore2 = scan.nextLine();

toReturn[2] = Integer.*parseInt*(tore2);

System.***out***.println("Column:");

String tore3 = scan.nextLine();

toReturn[3] = Integer.*parseInt*(tore3);

//System.out.println("print move");

Object from = (Object)(tore0 + tore1);

Object to = (Object)(tore2 + tore3);

*assertNotEquals*(from,to);

scan.close();

}

/\*\*

\* This method test move method work as accepted

\*/

*@SuppressWarnings*("static-access")

*@Test*

public void testMove() {

//System.out.println("tying to move "+ID+" from: "+i\_coordinat+","+j\_coordinat+" to: "+i+","+j);

//find the cell the player want to move to

Pawn testPawn = new Pawn(1,1,1,*PawnColor*.***BLACK***);

int i = 1, j = 1;

Board board = new Board();

Cell temp = board.*getCell*(1,1);

if (temp.getPawn\_on() == null && testPawn.move\_is\_legal(1,1)) { //check if the cell is empty

//make the move:

//set new coordinate for the pawn

testPawn.setI\_coordinat(i);

testPawn.setJ\_coordinat(j);

//update cell with this new pawn on

temp.setPawn\_on (testPawn);

//update the cell on the board

board.*setCell*(i,j,temp);

*assertNotNull*(temp.getPawn\_on());

*assertNotNull*(Board.*getCell*(i,j));

}

else {

*assertFalse*(false);

}

}

/\*\*

\* this method test if board get initialize with the correct number of pawns

\*/

*@SuppressWarnings*("static-access")

*@Test*

public void checkNumOfPawnsInBoard() {

Board testBoard = new Board();

*assertFalse*("size is not valid", testBoard.*getNumOfPawns*() == 25);

}

/\*\*

\* This method test if eating method work when the user try to eat opponent pawn

\*/

*@SuppressWarnings*("static-access")

*@Test*

public void testEating() {

int i = 1, j = 1;

Board board = new Board();

Cell temp = board.*getCell*(i,j);

temp.setPawn\_on(null);

Board.*setCell*(i,j,temp);

*assertNotNull*(Board.*getCell*(i,j));

*assertNull*(temp.getPawn\_on());

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| Actual Result | Expected Result | Description | Test ID |
| Questions added from JSON file | Questions added from JSON file | Test if the JSON file exist and get imported correctly to the game | 1 |
| User move was done correctly | User move was done correctly | Test the move the user want to make in his turn is ok to do | 2 |
| User move is pawns as accepted | User move is pawns as accepted | Test move method work as accepted | 3 |
| Number of pawns is OK | Number of pawns is OK | Test if board get initialize with the correct number of pawns | 4 |
| The pawn was eaten | The pawn was eaten | Test if eating method work when the user tries to eat opponent pawn | 5 |

1. טבלת SCRUM מצורפת להגשה.