**Navy Beach – The Board Game**

**Project Report**

**Project Title :** Navy Beach

**Course :** Software Project Management (CS521A)

**Team Name :** Challengers

**Team Details :**

|  |  |  |
| --- | --- | --- |
| Role | Student ID | Name |
| Developer | 17188 | E Ramchander Reddy |
| Developer | 17939 | Surampudi Venkata Ganesh Swamy Narayana |
| Developer | 18450 | Swaroop Raj |
| QA | 18482 | Lakshmi kanth somavarapu |
| Marketing | 18260 | Avanish Kumar aleti |

**Vision Statement:**

Navy Beach will allow User to Play a Great game with the opponent. Since it is very simple and powerful game, Customers will play for sure at any time. The project vision of the desired outcome to be produced for business after successful project completion. This examines Compatibility, Capability, Efficiency and maintainability. So Profit can give much in for Stakeholders.

**Project Scope:**

1. Navy Beach is played with two players and one will be the opponent (Computer).
2. No other Future is included in this other than stated above
3. Extra needs should be handled separately with extra cost and effort.

**Specifications:**

* **Functional Requirements:**

Rules of Play Beginning

Your Marines (blue) have captured the south beach on a small island. However, the enemy army (brown) still holds the beach on the north side of the island. Your mission is to cross the island and take the enemy's command bunker.

1. You can win the game by either blowing up your opponent's bunker, or killing all their movable pieces, or your opponent can not move (stuck behind his own mines).
2. The defending brown army gets to move first.
3. Movable pieces can only move one position at a time. They can only left or right or forward and backward. Diagonal moves are not allowed. Multiposition moves are not allowed.
4. Like Western Chess, you move your piece and touch the opponent's piece to attack it. Since mines and the bunker cannot move, they cannot attack.
5. During an attack, both sides must state the rank of their piece.
6. During an attack if both pieces are the same rank, then both pieces are removed from the game.
7. If a Sapper attacks a bomb, only the bomb is removed from the game.
8. If a Sapper attacks the bunker, then the game is won by the Sapper
9. Play the Game.
10. Check Placement of the pieces.
11. Exit.
12. Keep track of remaining pieces.

* **Interface Requirements:**
* Software will run on Java Runtime Environment (JRE).
* **Physical Environment Requirements:**
* Software will run on any system with Java Virtual Machine installed.

**Non-Functional Requirements:**

* **Users and Human Factor Requirements:**
* System supports any level of user with basic knowledge of computers.
* **Documentation:**
* The Information will be provided on Software Installation and running, so any user should be able to run the software.
* **Data Requirements:**
* Every turn should be stored to ensure game consistency.

**Implementation Techniques:**

Top-down is the implementation technique used in project. Code can be reused. All of the techniques are documented.

**Modules:**

* Start Game
* Teams
* Player Selection
* Sound
* Map Creation
* Status

**Project Description:**

Navy Beach is the game played on Board. The Board is 11 positions wide. The game is played by two Players; the opponent’s army takes the other side of the board. And the Team must place a piece only on one side of the board initially.

Before Play begins, each player arranges a piece on their side. Since no piece can swim they must not swim across the lake on the island. The Marine blue has to capture the south small island, and the enemy army holds beach on the north side.

In order to win we have to cross the island and take the enemy bunker. We can win the game by killing all the pieces or destroying the enemy bunker.

Each army consist of following pieces:

|  |  |
| --- | --- |
| **Rank Order (Piece)** | **QTY** |
| General | 1 | |
| Colonels | 3 | |
| Majors | 3 | |
| Lieutenants | 8 | |
| Sergeants | 4 | |
| Mine Removing Sappers | 5 | |
| Marines | 9 | |
| Non-Movable Mines | 6 | |
| Non-Movable Bunker | 1 | |

**Note:** In our project we consider Piece as of Qty 1.

After the pieces have been positioned, the game proceeds in a series of rounds. In each round each player takes a turn to announce a target square. The defending Brown army gets to move first and the piece can only move one position at a time. The pieces can only move left or right and forward or backward.

**Note:** The pieces cannot move in Diagonal and multiple moves are not possible.

During the attack a piece must state a rank of their piece. If both pieces are of the same rank during an attack then both the pieces are to be removed from the game.

If all pieces are killed or the bunker destroyed or the pieces stuck behind their own mines the game is over and the opponent wins.

**Data Flow Diagram for system modules:**

Start Game

Team Blue

Team Brown

Make sure Teams are ready

Status

(won or lose)

Stays in the same loop Until Task Done

Display Positions

After Shifting

Players Turn upon Selection

**System Block Diagram with Modules and Interfaces**

**Team 1**

**Main Module**

**New Game**

**Exit Game**

**Team 2**

Team 1

moves

**Yes**

**New/End**

**Game**

**Check if Team 1 won**

**Check if Team 2 won**

**No**

**No** **Yes**

**Team 2 moves**

**Schedule and Effort Charts (non-gantt)**

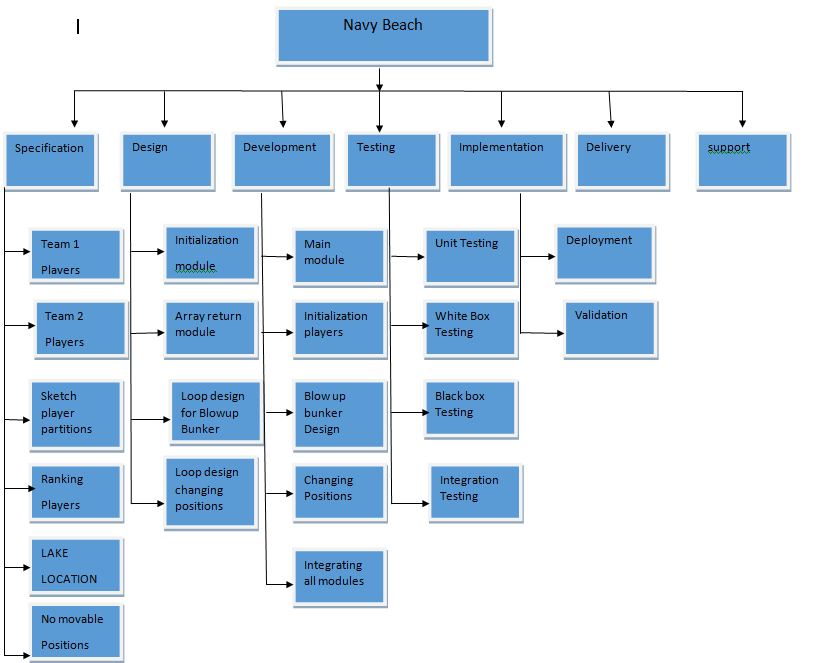
|  |  |
| --- | --- |
| **Project Activities** | **Effort days** |
| **Specifications** | **6** |
| **Design Analysis** | **13** |
| **Coding Implementation** | **20** |
| **Test Cases** | **13** |
| **Test** | **13** |
| **Delivery** | **7** |
| **Support** | **7** |

**Work Breakdown Structure (WBS):**

The work break down structure provides project management process like estimation, Scheduling, risk. It provides framework about the project status and progress report.

Top most level represents final project. The work package defines work, duration and cost. It should be independent of others. All the elements of WBS don’t need to be defined to same level.

**WBS:**



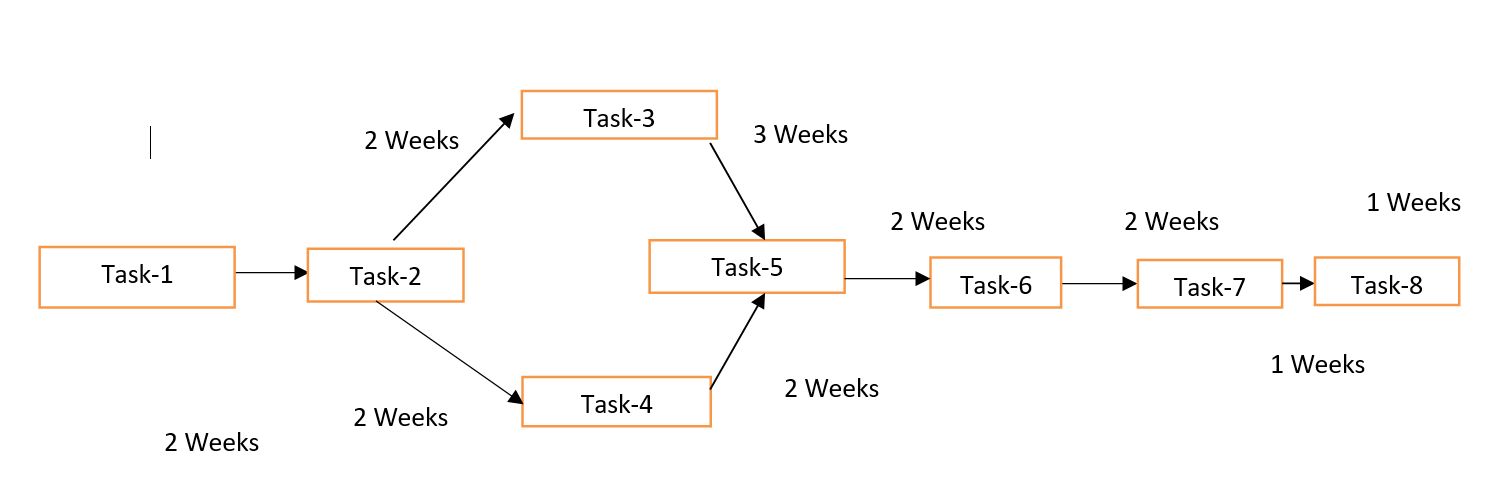
**Project Estimation:**

Duration : 11 Weeks

Resources : 8

Total Cost : $ 200,000

**PERT Analysis:**



|  |  |
| --- | --- |
| Task 1: Specifications | 2Weeks |
| Task 2: Design Analysis | 2Weeks |
| Task 3: Coding and Implementation | 3Weeks |
| Task 4: Test case preparation | 2 Weeks |
| Task 5: Testing | 2 Weeks |
| Task 6: Implementation | 2 Weeks |
| Task 7: Delivery | 1 Weeks |
| Task 8: Project Support | 1 Weeks |

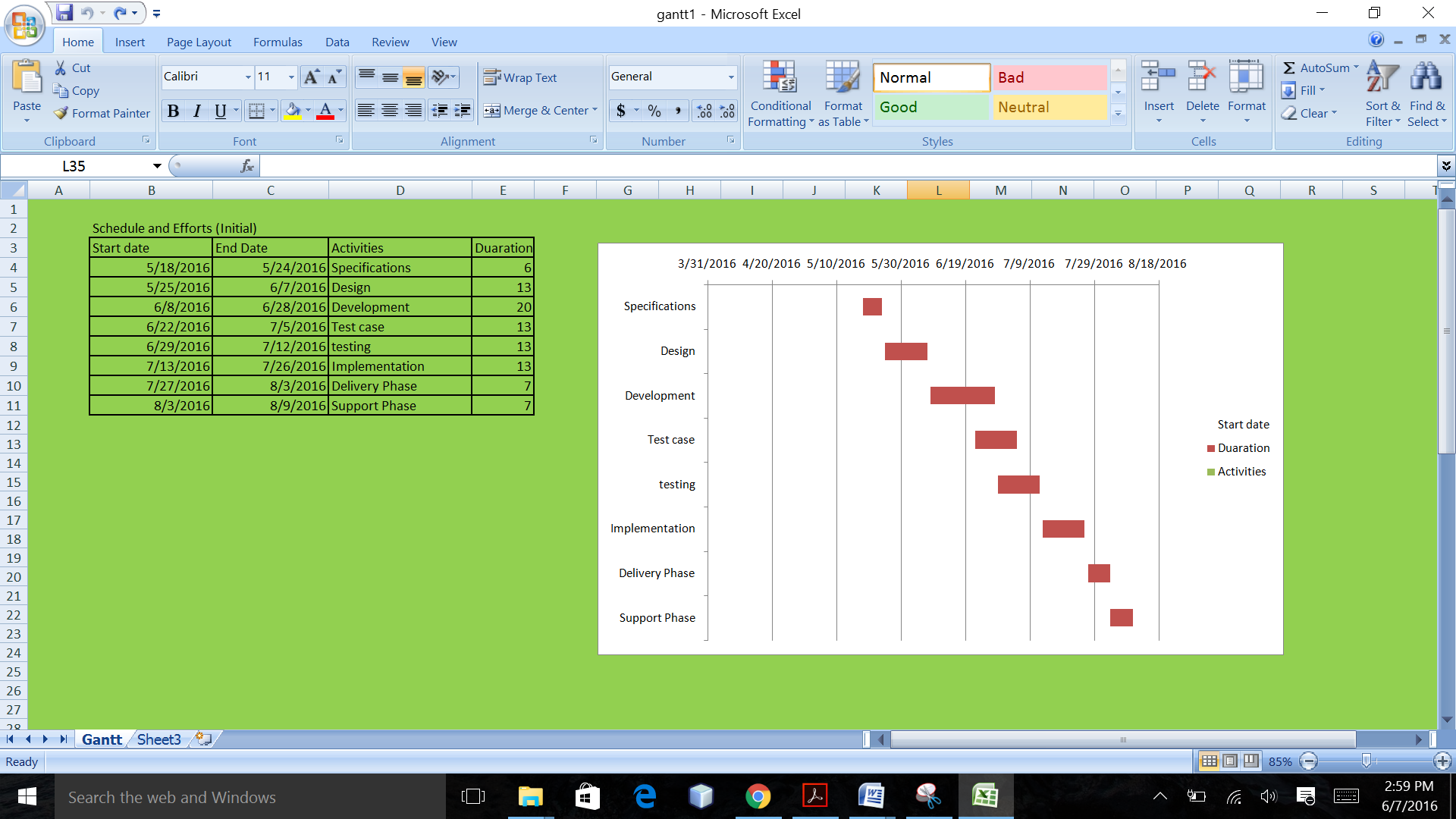
In pert activities are shown as a network of precedence relationship in which successive events are joined by arrows

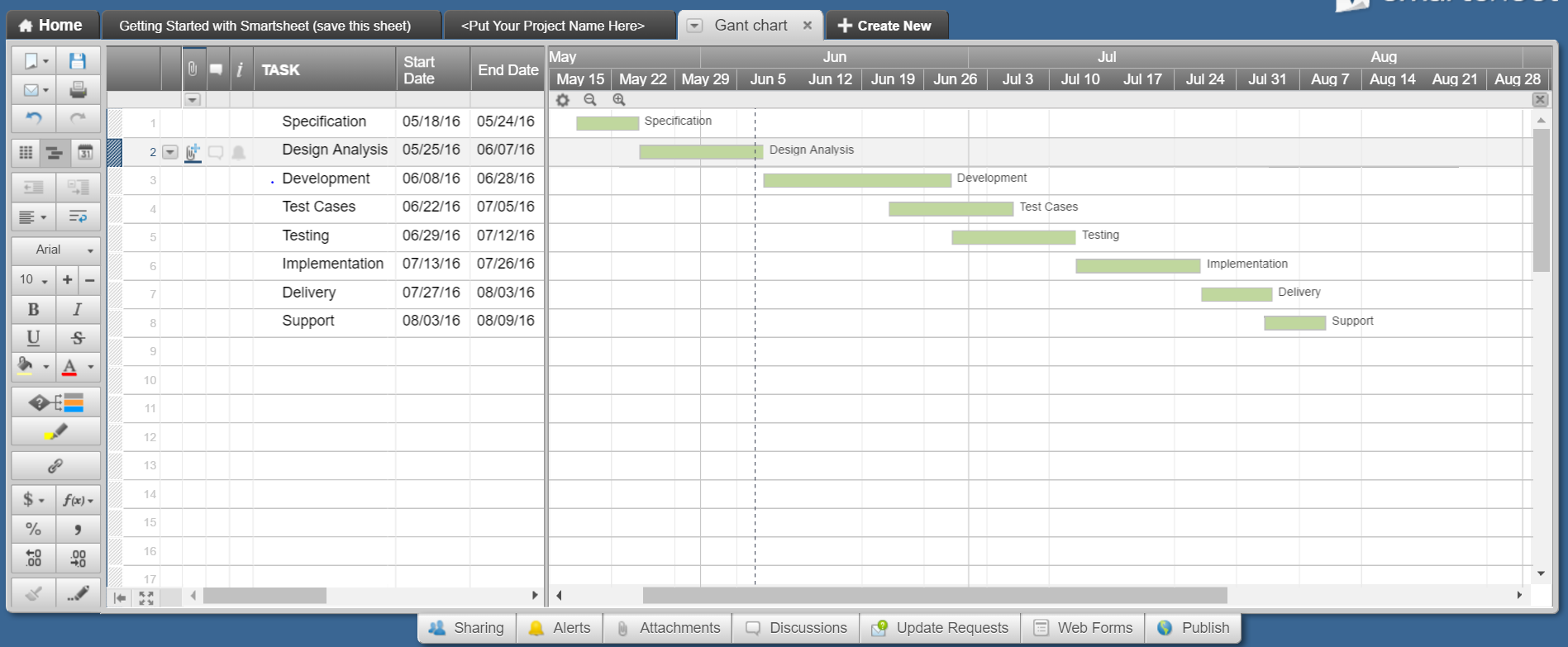
1. Probabilistic activity times
2. Optimistic time, pessimistic time, most likely time event oriented diagram.

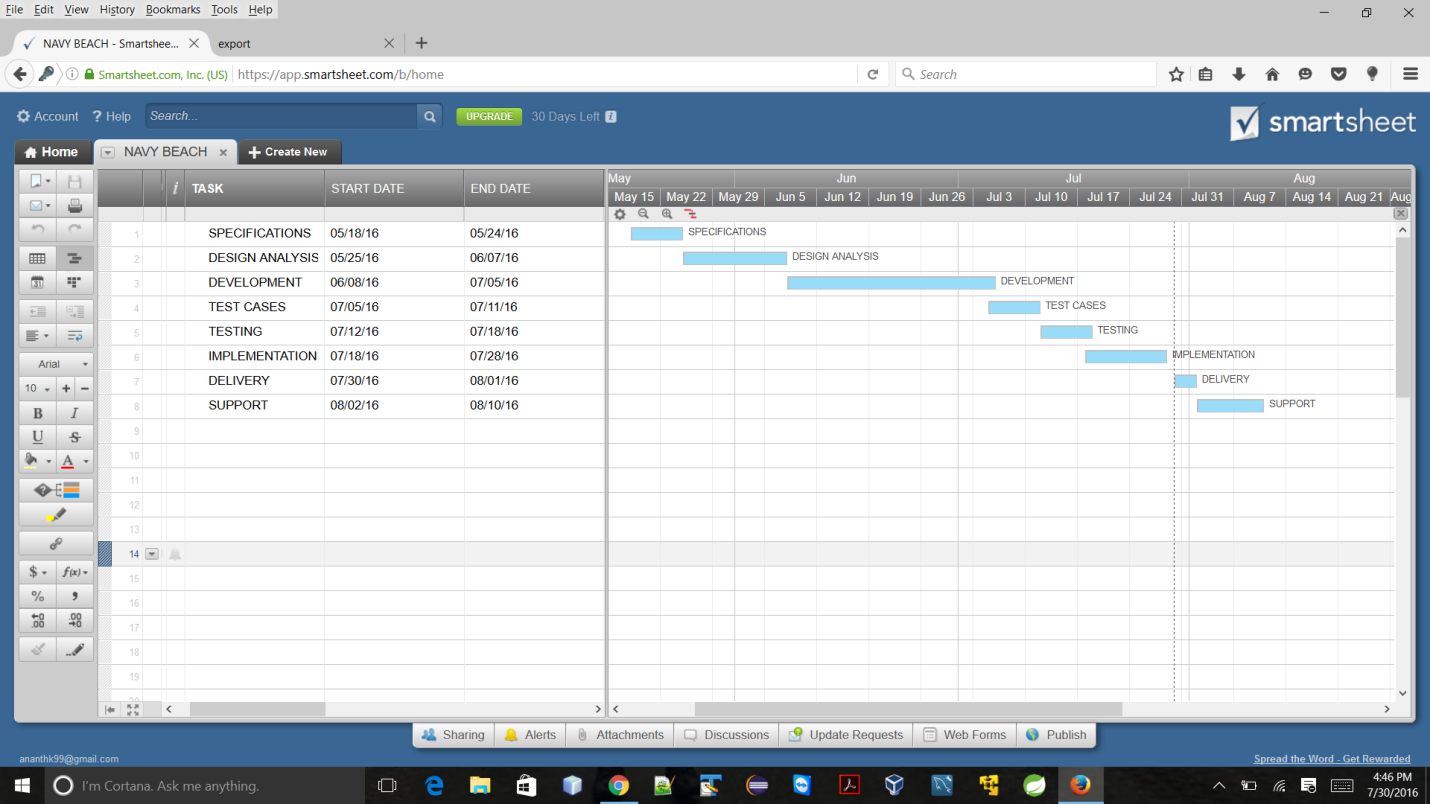
For non-repetitive jobs where the time and cost estimates tend to be quite uncertain. This technique uses probabilistic time estimates.

PERT is used to assist in project scheduling similar to CPM however, pert assumes that activity duration and random variables

**Gantt Chart:**

****

****

****

The PERT chart is sometimes preferred over the Gantt chart, another popular project management charting method, because it clearly illustrates task dependencies. On the other hand, the PERT chart can be much more difficult to interpret, especially on complex projects. Frequently, project managers use both techniques.

**Critical path**

2 Weeks

3 Weeks

Task-3

1 Weeks

2 Weeks

2 Weeks

Task-8

Task-1

Task-7

Task-6

Task-5

Task-2

1 Weeks

2 Weeks

2 Weeks

Task-4

2 Weeks

The PERT/CPM procedure

There are the six stages common to both PERT and CPM:

1) Define the project and specify all activities or tasks.

2) Develop the relationships amongst activities. Decide upon precedence’s.

3) Draw network to connect all activities

4) Assign time and/or costs to each activity.

5) Calculate the longest time path through the network: this is the “critical path”.

6) Use network to plan, monitor and control the project.

The Longest path is shown in red color arrow marks

**Assumptions:**

1. User has an Internet Browser which supports Java (Google Chrome, Firefox, and Internet Explorer 11).
2. User has Latest Graphic card.

**Tools and Computing Environment:**

* Developed using Java Environment
* Will run on any Operating System with Java Virtual Machine Installed.

**Software Life Cycle:**

* Extensive Testing will be done to find Defects or Errors.
* If an Error occurs after the release new updates may be released.
* There will be a final version which will have a lifetime Warranty.

**Test Cycles:**

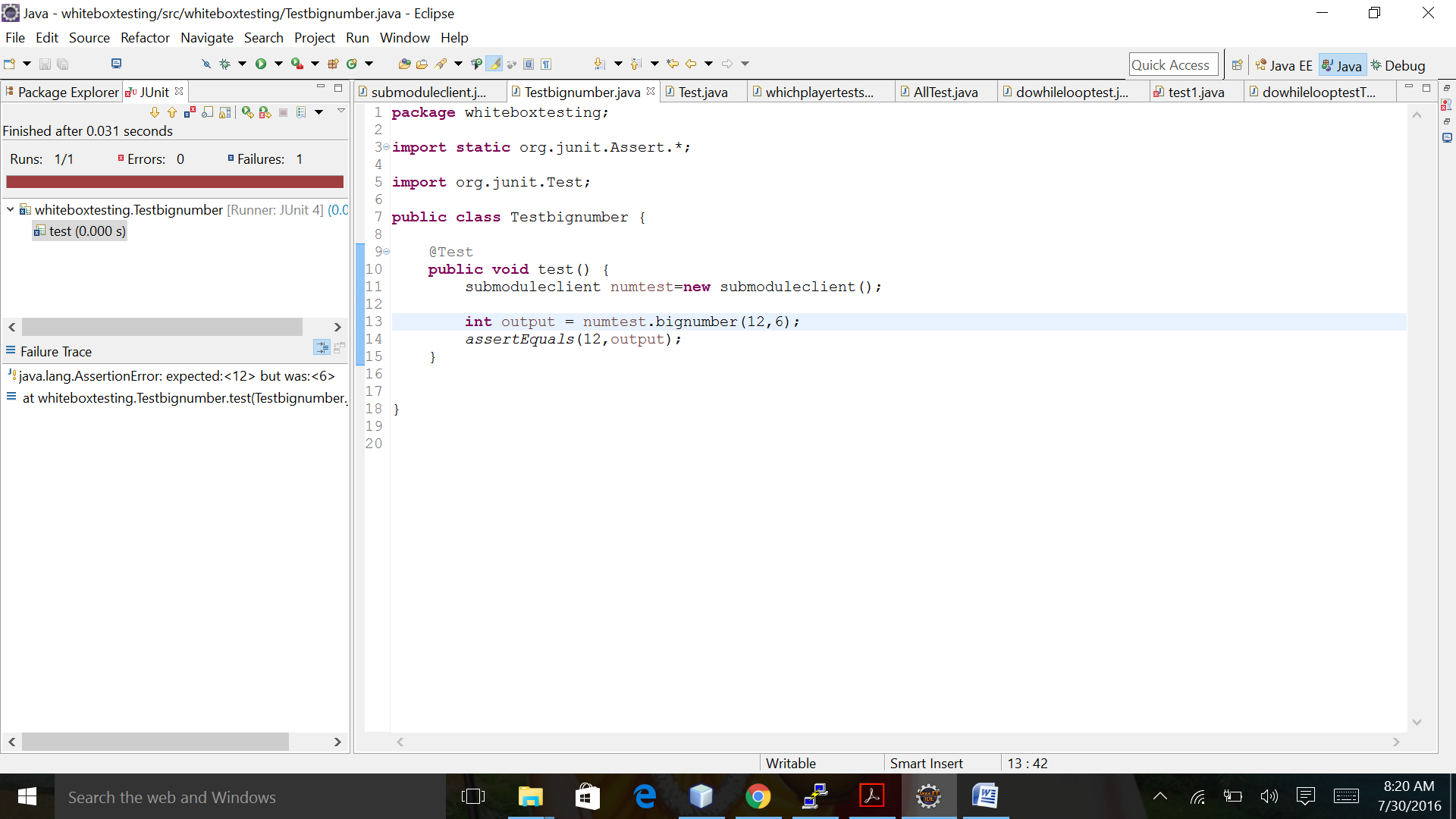
The objectives of the cycles are to test the critical defects and identify objects blocking. It is expected to do work around to get all the scripts unit test has first module and then integration test.

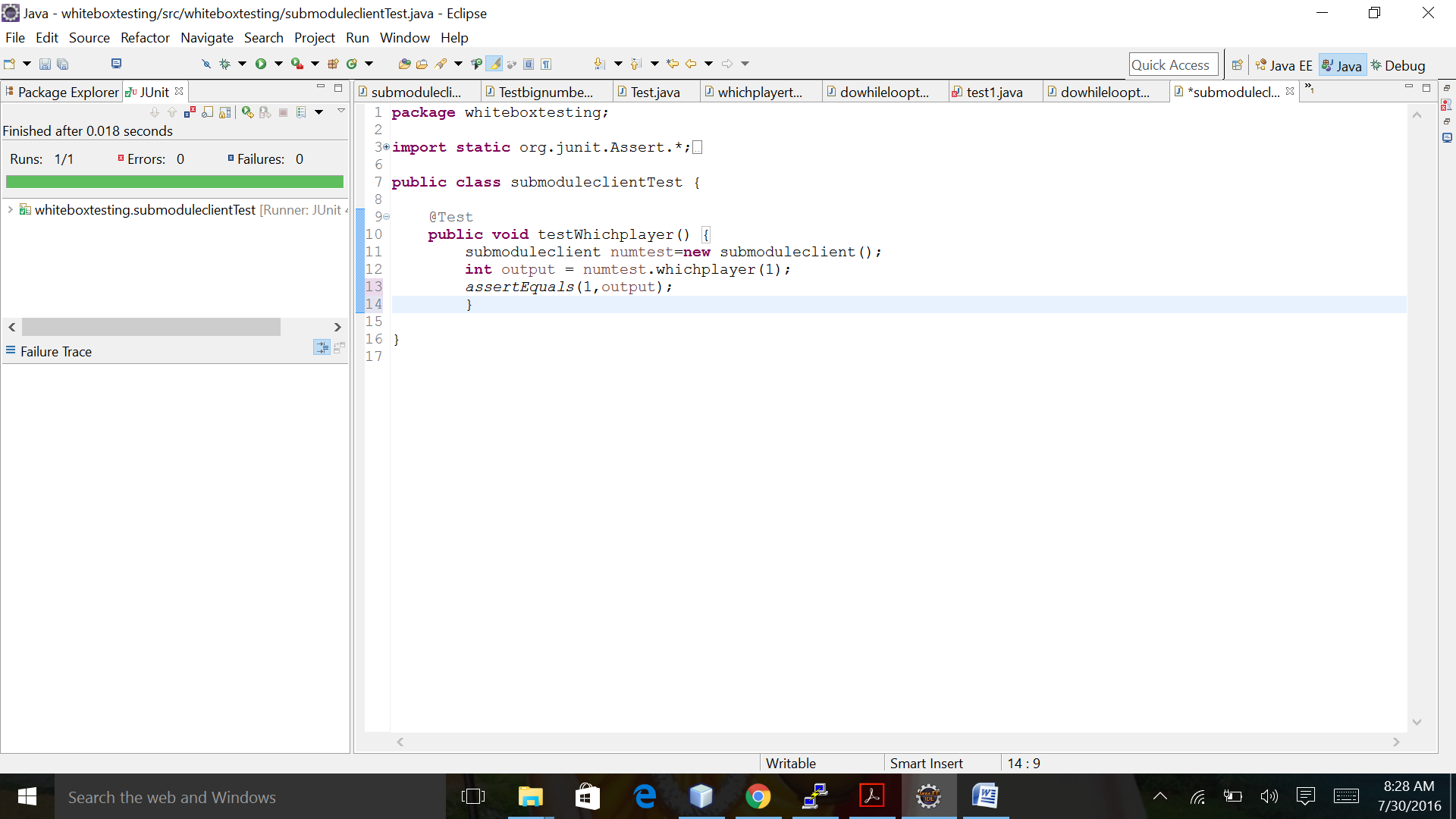
**Scope and levels of Testing:**

The purpose of this test is to make sure critical defects are removed before the next levels of testing can start.

There are two approaches when creating software testing

* **Black box Testing:** Also referred to as functional testing. It focuses on testing the internal mechanism in the program.
* **White box Testing:** It is a form of structural testing that is also called clear or open box testing. It is the opposite of black box testing.





**Test Environment:**

* Find issue and errors
* Correct Errors
* Unit Testing
* Regression Testing

**Quality Assurance:**

The development team will check frequently for assured quality of project. The Test plan will be consulted for specific tests but in additional unit test will be introduced frequently during development to assure the product is working as per the document requirements.

**Unit Testing:**

It is the smallest testable component of a program. The modules are made up of units it isolates small section of program and test the individual parts to prove work correctly.

public class submoduleclient{

public int bignumber(int s , int count)

{

int x=0;

if (s == 127) {

//battle\_start();

System.out.println("\n");

System.out.println("Blue own the game");

x=s;

}

else {

System.out.println(" More Sappers left");

x=count;

//System.out.println(count);

}

return x;

}

/////////////////////////Test case//////////////////

package whiteboxtesting;

import static org.junit.Assert.\*;

import org.junit.Test;

public class Testbignumber {

@Test

public void test() {

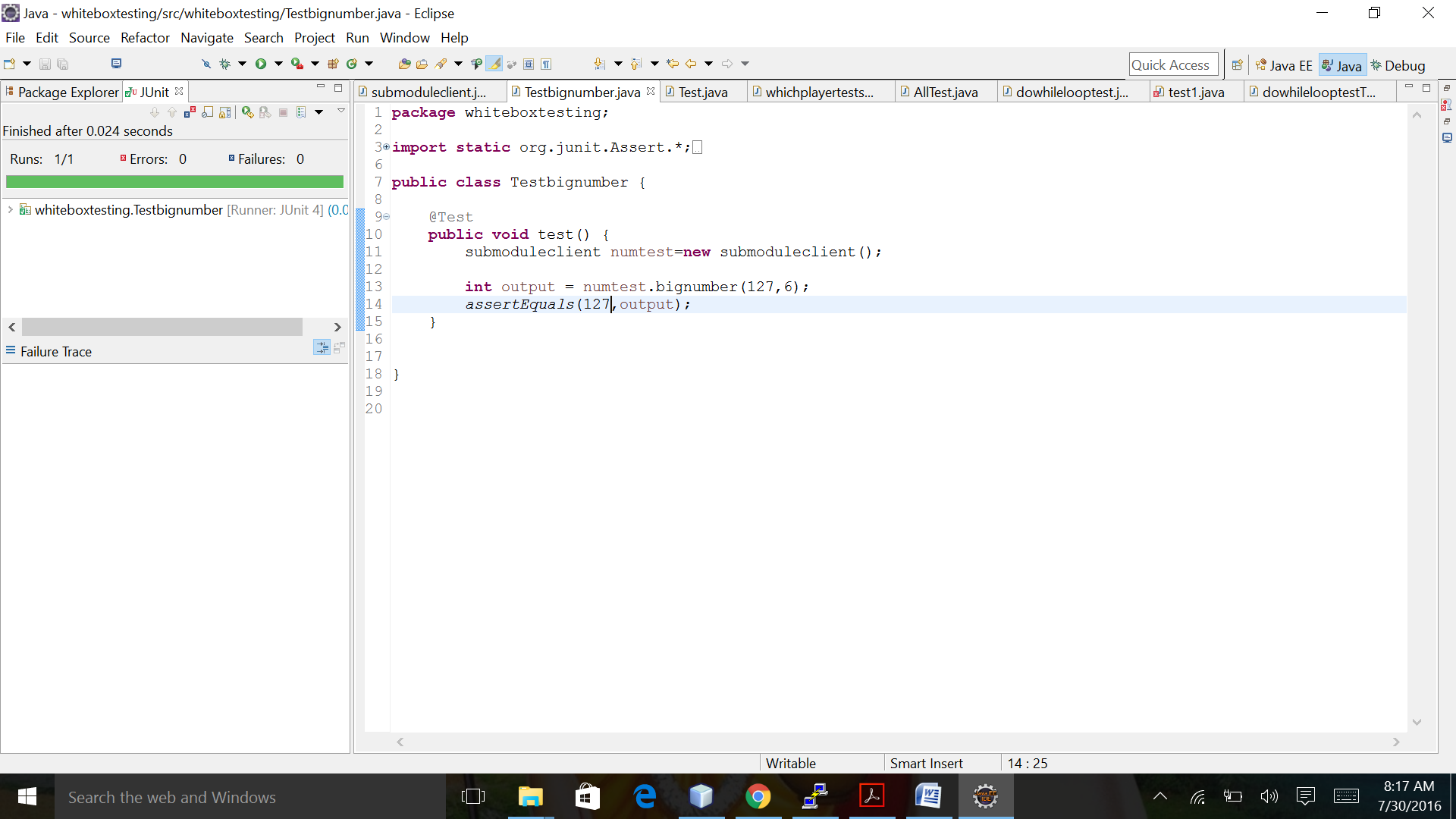
submoduleclient numtest=new submoduleclient();

int output = numtest.bignumber(127,6);

assertEquals(127,output);

}

}



**Integration Testing:**

It combines individual modules and test as group. These test cases take modules that have been unit tested. The output is which readies final layer of testing.

**System Testing:**

It is the Final layer of software testing. It is conducted once the system has been integrated. It falls within the black box Testing. It covers Usability, Reliability, Maintenance, Compatibility and Performance.

**Security:**

* Software Warranty will be void if the code is modified.
* No implemented Security.

**Risk Management:**

* New updates released over the internet to reduce the cost of failures.
* Extensive Testing to reduce Errors and Failures.

**Maintenance Plan:**

* A new version will be released.
* Maintenance will be done given errors occur after the final release.

**Implementation:**

1. Modules
2. Combine all the code in Java package to implement in the production.

**Schedule Time Line and Dependencies:**

1. Schedule and Effort Charts
2. Multiple Updates
3. Critical Path
4. Gantt Charts

**Note:** Project Design, Coding and Implementation is critical path.

**Version Control:**

Version Tracking:

Developers may wish to compare today’s version. Since version control system keeps track of every version of the software. Any problems that arose from a change can then be followed up by an examination of who made the change and reasons they gave for making the change.

**Team Coordinating:**Team carries out the Resource development. Version control is central to coordinating teams of contributors. Other contributors work on their own copies of the same resources at same time, Typically in open source projects version control system allow anyone to read and copy the project resources but only authenticated users know and are allowed to update the source code.

**Testing Cycle:**

* Find Issue and Errors
* Correct Errors
* Unit Testing
* Regression Testing
* Test new Features

**Quality Assurance:**

The development team will check frequently for assured quality of project. The Test plan will be consulted for specific tests but in additional unit test will be introduced frequently during development to assure the product is working as per the documentation requirements.

**Issues Priority:**

* Critical

1. Issues that cause the programs to crash.
2. Issues that cause the program to crash.
3. Issues that prevent game from finishing.

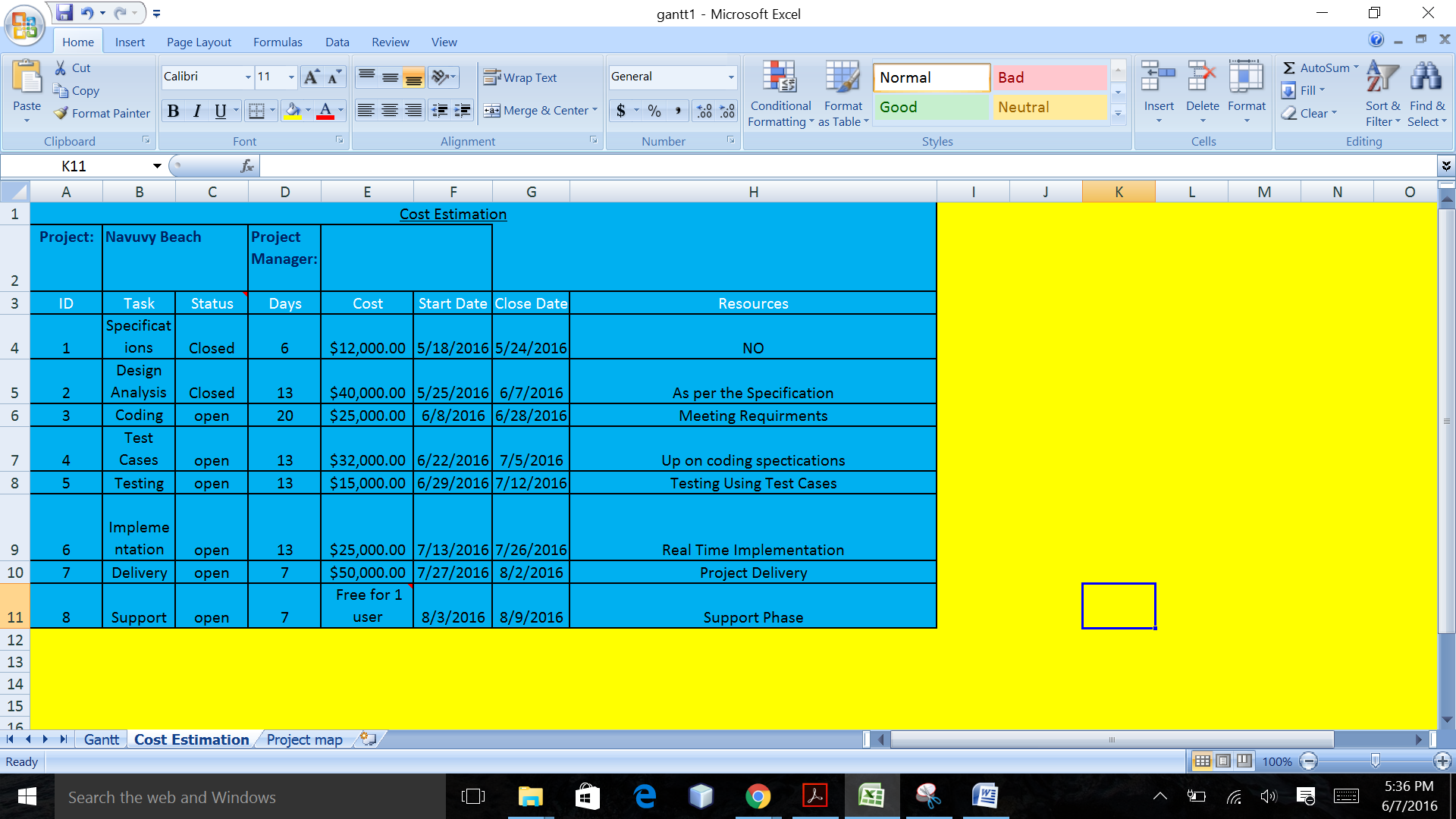
* Intermediate

1. Issues that cause the game to enforce rules.

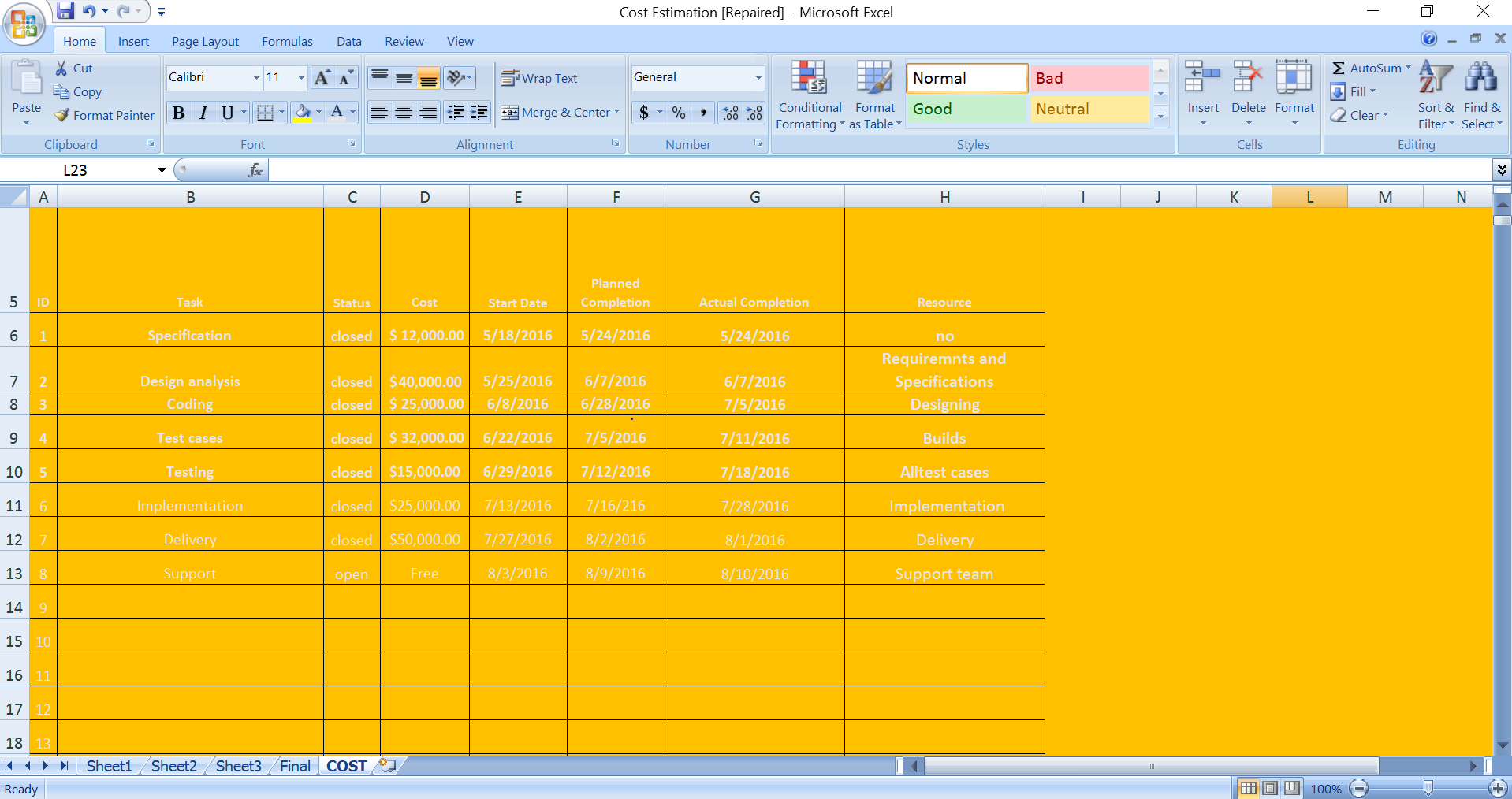
* Low

1. Graphical glitches
2. Advance features that are not specific in the requirements

**Cost Estimation:**



After Completing The Project



**Test Completion Criteria:**

Testing should be stopped when it meets the completion criteria. There are no known critical bugs. As testing is never ending process we can never assume 100% testing has been done. Coverage of code and requirements reached as specified.

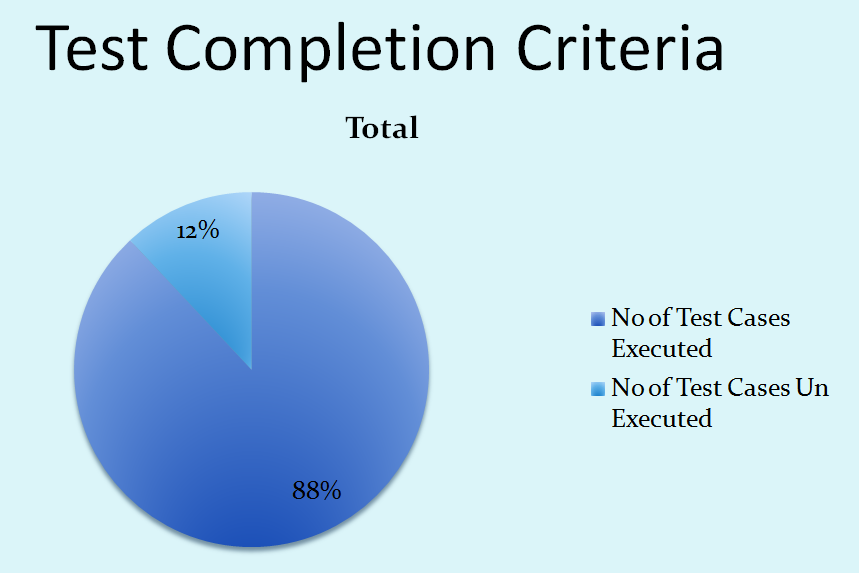
Test completion status = NO of Test Cases Executed

--------------------------------------------------------- X 100%

Total no of Test cases

= (22/25) X 100%

= 88%



Which player selection

**public** **int** whichplayer(**int** i) {

**int** player1=1;

**int** player2=2;

**int** x1=0;

// Am I player 1 or 2?

**if** (player1 == i)

{

x1=player1;

}

**else** **if** (player2 == i)

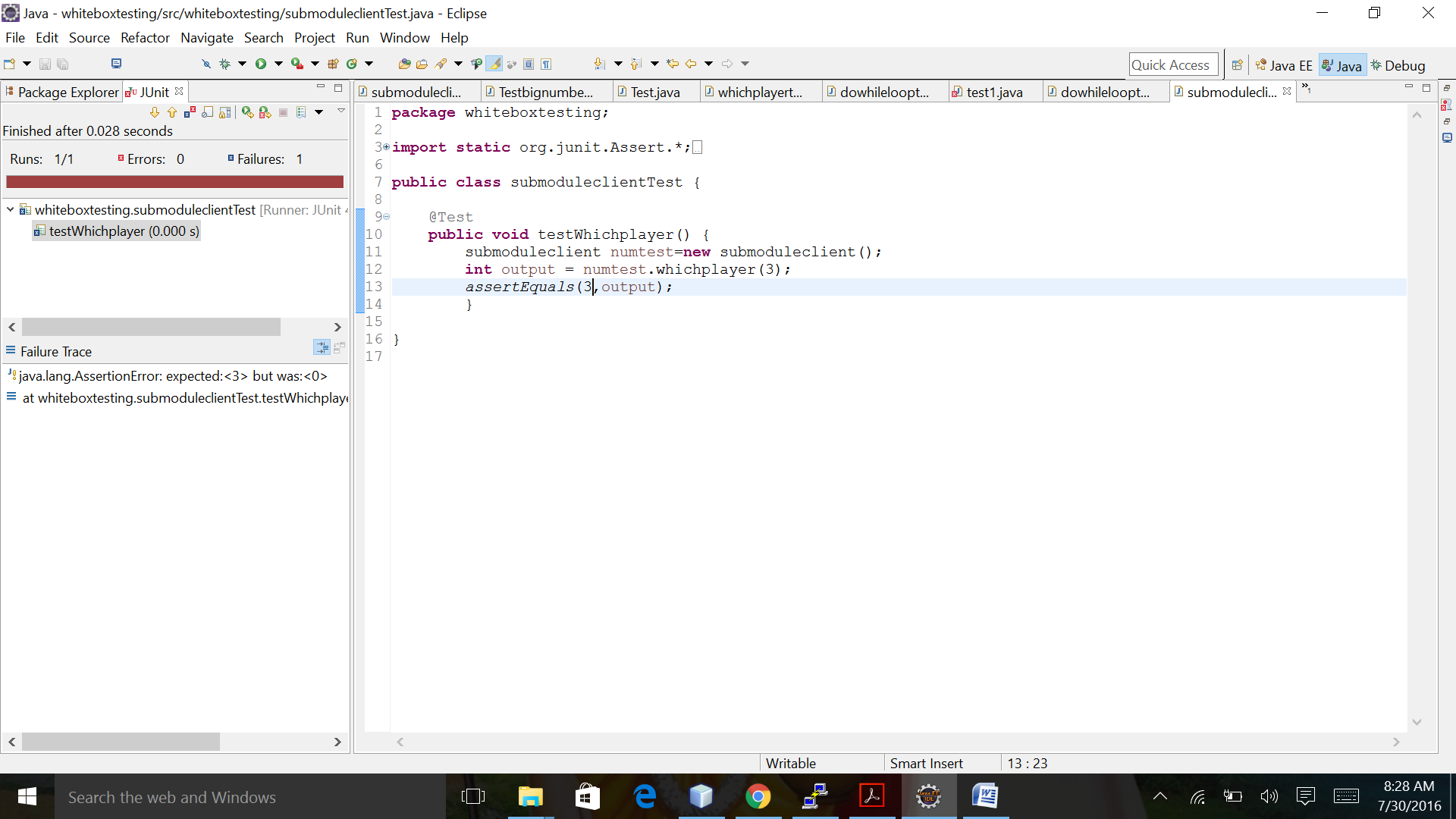
{

x1=player2;

}

**return** x1;

}



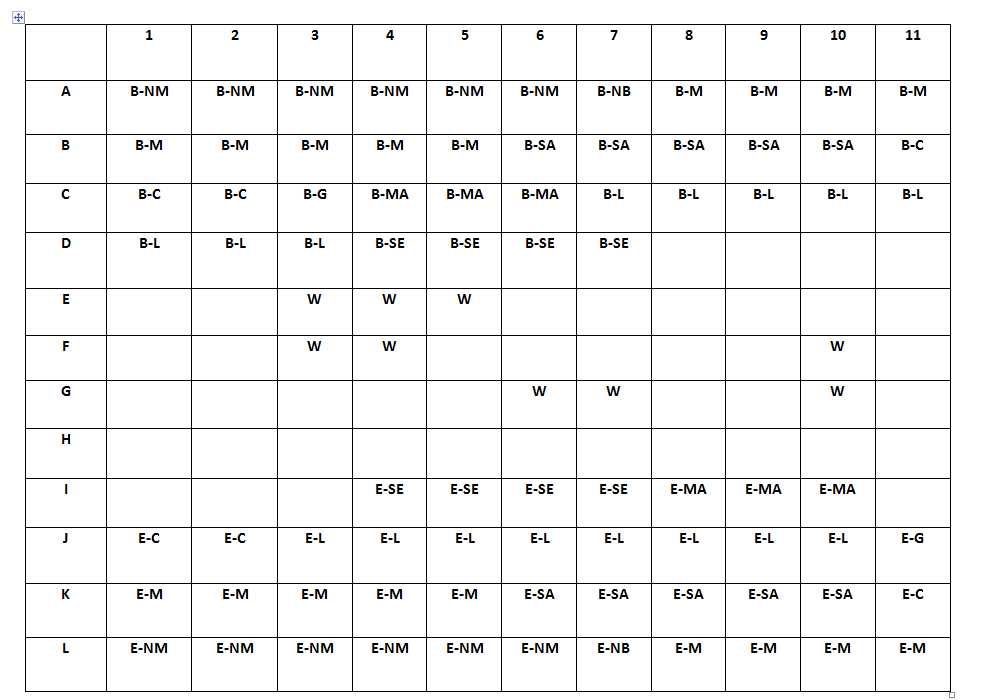
**Milestones:**

|  |  |  |
| --- | --- | --- |
| **Milestone no:** | **Milestone** | **Completion Date** |
| M1 | Specification | 5/24/2016 |
| M2 | Design Analysis | 06/07/2016 |
| M3 | Coding Implementation | 06/28/2016 |
| M4 | Test Cases | 07/5/2016 |
| M5 | Testing | 07/12/2016 |
| M6 | Implementation | 07/26/2016 |
| M7 | Delivery | 08/03/2016 |
| M8 | Project Support | 08/09/2016 |

Milestone:

|  |  |  |
| --- | --- | --- |
| **Milestone no:** | **Milestone** | **Completion Date** |
| **M1** | **Specification** | **5/24/2016** |
| **M2** | **Design Analysis** | **06/07/2016** |
| **M3** | **Development** | **07/05/2016** |
| **M4** | **Test Cases** | **07/11/2016** |
| **M5** | **Testing** | **07/18/2016** |
| **M6** | **Implementation** | **07/28/2016** |
| **M7** | **Delivery** | **08/01/2016** |
| **M8** | **Project Support** | **08/10/2016** |

NAVY BEACH ARMY POSITIONS:



**Appendix:**

First Blue team will start the game

Blue team army will try to blow the brown team non movable bunkers. If blue team fails to hit next brown team will attack on blue. If both fail to blow now army people go for fighting by shifting they positions. If no moves are available last move team will win the game.

Symbolic representation:

NM:NON MOVABLE MINES

NB:NON MOVABLE BUNKERS

M :MARINES

SA:SAPPERS

MA:MAJORS

SE:SERGEANTS

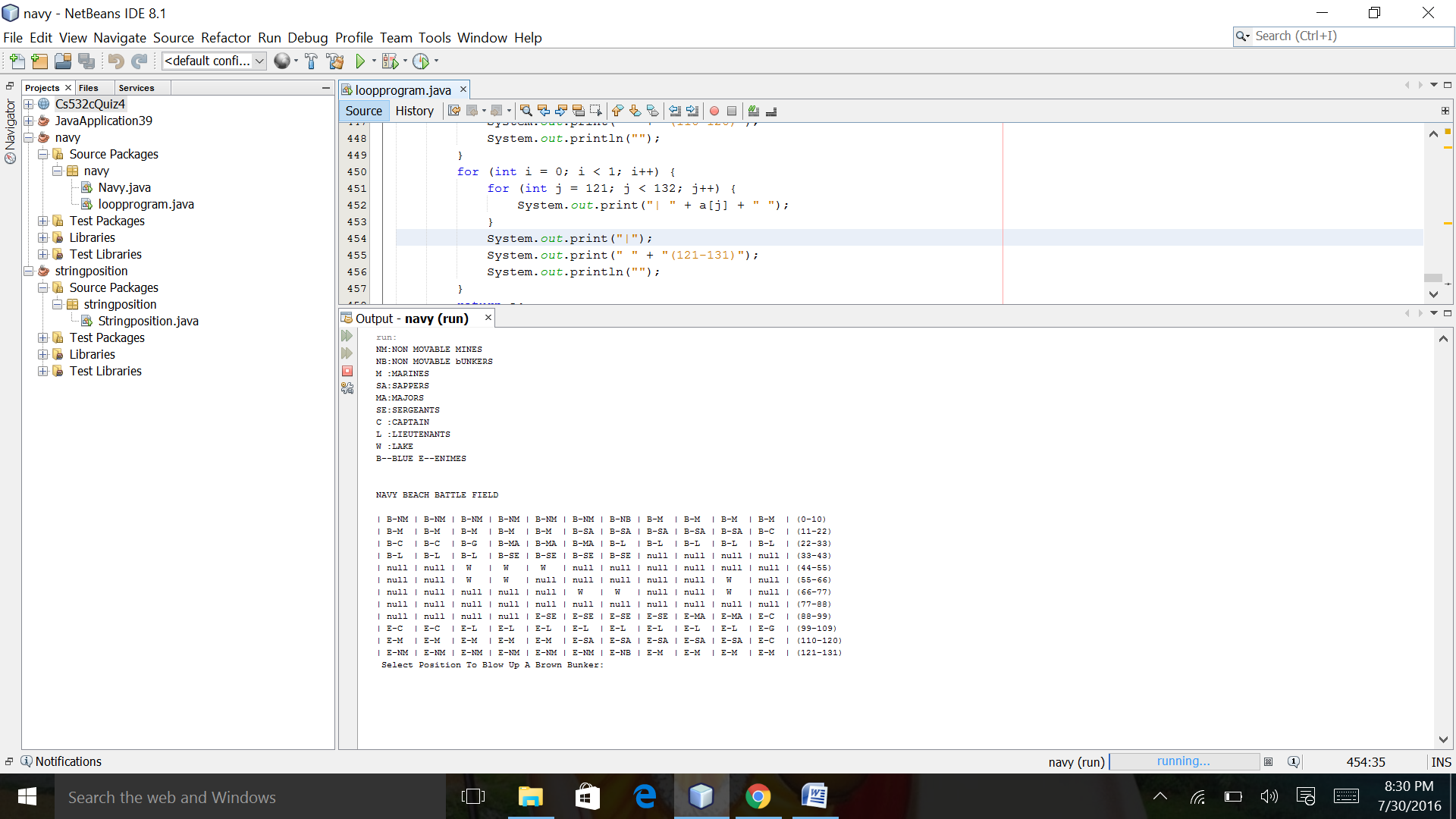
C :CAPTAIN

L :LIEUTENANTS

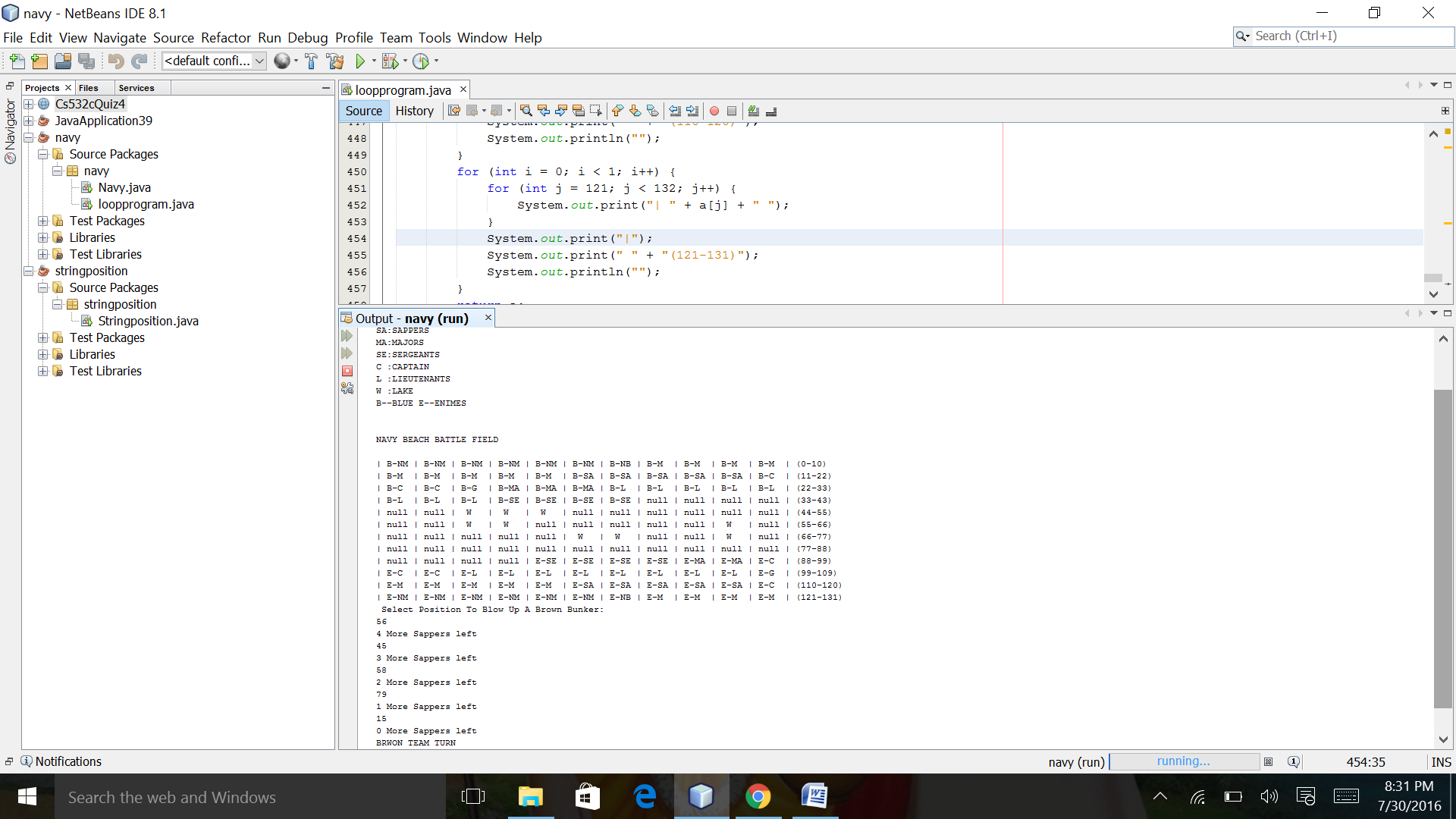
W :LAKE

B--BLUE E--ENIMES(Brown)

Start the game:

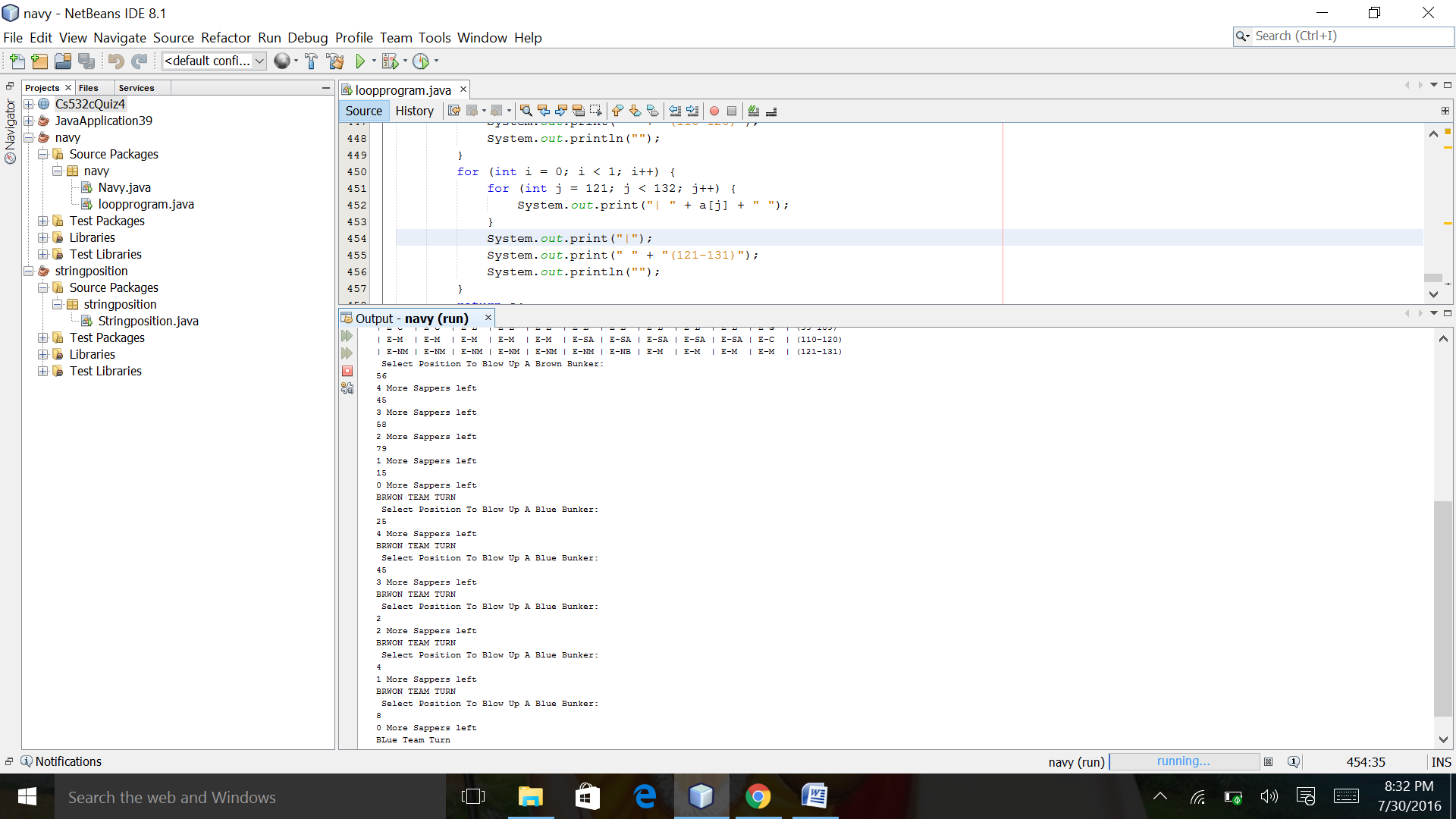


Blue team will try to blow brown team bunkers



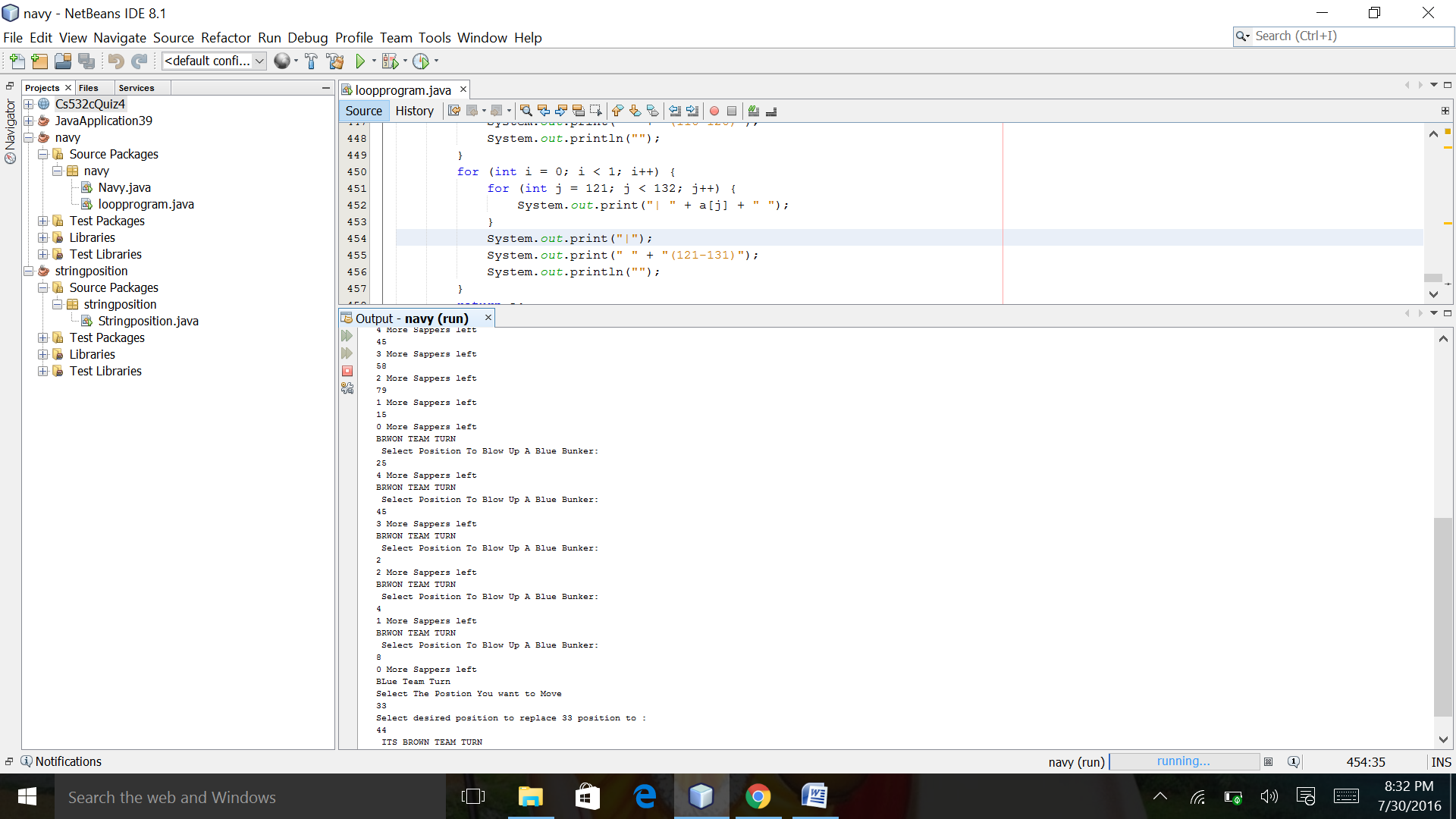
Blue team fails to attack on bunkers

Now brown team turn

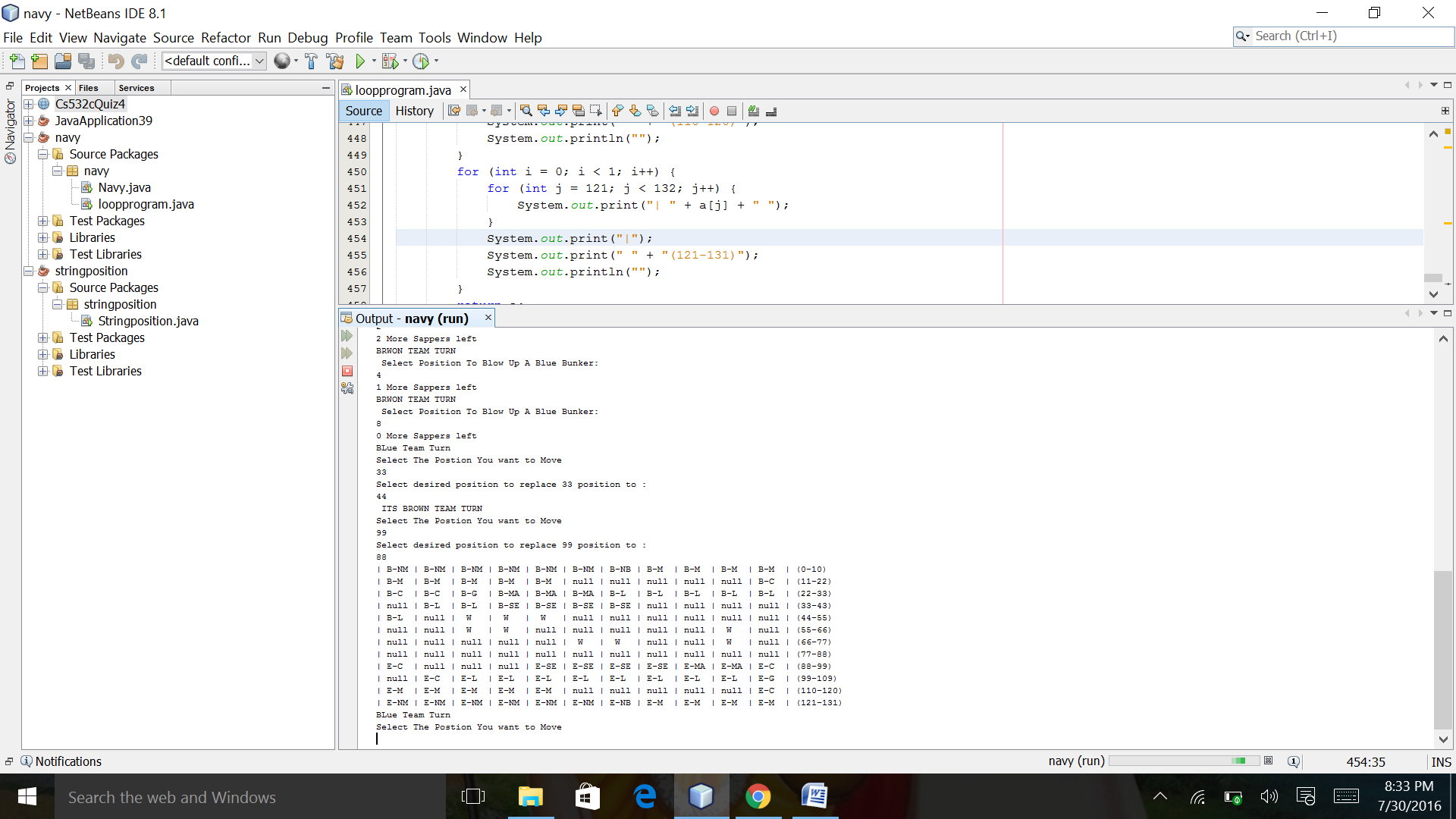


Brown team fails to attack blue team

Now blue team turn. Fighting starts



In the above screen shot you can see how the positions are shifted



Now blue team will play

Until the movable positions are available. If they struck the opposite team will win.

///////////////////////////Code For Navy Beach////////////////////

package navy;

import java.util.Random;

import java.util.Scanner;

/\*\*

\*

\* @author venka

\*/

public class loopprogram {

public static void main(String[] args) {

System.out.println("NM:NON MOVABLE MINES");

System.out.println("NB:NON MOVABLE bUNKERS");

System.out.println("M :MARINES");

System.out.println("SA:SAPPERS");

System.out.println("MA:MAJORS");

System.out.println("SE:SERGEANTS");

System.out.println("C :CAPTAIN");

System.out.println("L :LIEUTENANTS");

System.out.println("W :LAKE");

System.out.println("B--BLUE " + "E--ENIMES ");

System.out.println("");

System.out.println("");

System.out.println("NAVY BEACH BATTLE FIELD");

System.out.println("");

String[] b = new String[132];

int c = 0;

b = battle\_start();

System.out.println(" Select Position To Blow Up A Brown Bunker: ");

Scanner i = new Scanner(System.in);

//int x = i.nextInt();

int count = 0;

int count1 = 0;

int moves=0;

///////////////Bullet Attack ////////

do {

int s = i.nextInt();

//randomint(1,10);

if (s == 127) {

battle\_start();

System.out.println("\n");

System.out.println("Blue own the game");

break;

} else {

System.out.println(4-count+ " More Sappers left");

count++;

if(count==5){

b[16]=null;b[17]=null;b[18]=null;b[19]=null;b[20]=null;

}

//System.out.println(count);

}

} while (count < 5);

do {

System.out.println("BRWON TEAM TURN");

System.out.println(" Select Position To Blow Up A Blue Bunker: ");

int s1 = i.nextInt();

//randomint(1,10);

if (s1 == 6) {

battle\_start();

System.out.println("\n");

System.out.println("Brown own the game");

break;

} else {

System.out.println(4-count1+ " More Sappers left");

count1++;

if(count1==5){

b[115]=null;b[116]=null;b[117]=null;b[118]=null;b[119]=null;

}

//System.out.println(count);

}

}while (count1 < 5);

//////////

do {

System.out.println("BLue Team Turn");

System.out.println("Select The Postion You want to Move");

int x = i.nextInt();

String fil=b[x];

//b[x]=null;

System.out.println("Select desired position to replace "+x+ " position to : ");

int y = i.nextInt();

//randomint(1,10);

if (b[x] == null) {

System.out.println("\n");

System.out.println("Postion is empty please select other postion");

}

else if(b[x] !=null) {

if(b[y]!=null){

System.out.println("Position is not empty");

}

else{

b[y]=fil;

b[x]=null;

}

}

/////////////////////Now Enemy term///////////////

System.out.println(" ITS BROWN TEAM TURN ");

System.out.println("Select The Postion You want to Move");

int x1 = i.nextInt();

String fill=b[x1];

//b[x]=null;

System.out.println("Select desired position to replace "+ x1 + " position to : ");

int y1 = i.nextInt();

//randomint(1,10);

if (b[x1] == null) {

System.out.println("\n");

System.out.println("Postion is empty please select other postion");

}

else if(b[x1] !=null) {

if(b[y1]!=null){

System.out.println("Position is not empty");

}

else{

b[y1]=fill;

b[x1]=null;

}

}

/\* for(int z=0;z<110; z++){

if(b[z+10].equals(b[z])){

b[z]=null;

b[z+10]=null;

}

}\*/

for (int f = 0; f < 1; f++) {

for (int j = 0; j < 11; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(0-10)");

System.out.println("");

}

for (int f = 0; f < 1; f++) {

for (int j = 11; j < 22; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(11-22)");

System.out.println("");

}

for (int f = 0; f < 1; f++) {

for (int j = 22; j < 33; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(22-33)");

System.out.println("");

}

for (int f = 0; f < 1; f++) {

for (int j = 33; j < 44; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(33-43)");

System.out.println("");

}

for (int f = 0; f < 1; f++) {

for (int j = 44; j < 55; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(44-55)");

System.out.println("");

}

for (int f = 0; f < 1; f++) {

for (int j = 55; j < 66; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(55-66)");

System.out.println("");

}

for (int f = 0; f< 1; f++) {

for (int j = 66; j < 77; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(66-77)");

System.out.println("");

}

for (int f = 0; f < 1; f++) {

for (int j = 77; j < 88; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(77-88)");

System.out.println("");

}

for (int f = 0; f < 1; f++) {

for (int j = 88; j < 99; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(88-99)");

System.out.println("");

}

for (int f = 0; f< 1; f++) {

for (int j = 99; j < 110; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(99-109)");

System.out.println("");

}

for (int f = 0; f < 1; f++) {

for (int j = 110; j < 121; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(110-120)");

System.out.println("");

}

for (int f = 0; f < 1; f++) {

for (int j = 121; j < 132; j++) {

System.out.print("| " + b[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(121-131)");

System.out.println("");

}

/\*for(int z=0;z<110; z++){

if(b[z+11].equals(b[z])){

b[z]=null;

b[z+10]=null;

}

} \*/

} while (moves < 5);

}

public static String[][] initBoard() {

String[][] shades = new String[100][100];

// print array in rectangular form

for (int r = 0; r < 100; r++) {

for (int c = 0; c < 100; c++) {

shades[r][c] = " ";//your value

}

}

return shades;

}

//////////bullet method///////////////

public static boolean bullet(int count) {

boolean x = false;

if (count == 0) {

x = false;

} else {

x = true;

}

return x;

}

////////////////////Random Number Generator///////////////

public static int randomint(int min, int max) {

Random rand = new Random();

int randomNum = rand.nextInt((max - min) + 1) + min;

return randomNum;

}

////////////////////////Battle Start//////////////////////

public static String[] battle\_start() {

String[] a = new String[132];

//a[0]=1;a[1]=1;a[2]=1;a[3]=1;a[4]=1;

//a[5]=1;a[6]=1;a[7]=1;a[8]=1;a[9]=1;a[10]=1;

// a[11]=2;a[12]=2;a[13]=2;a[14]=2;a[15]=2;a[16]=2;a[17]=2;a[18]=2;a[19]=2;a[20]=2;

//a[21]=3;a[22]=3;a[23]=3;a[24]=3;a[25]=3;a[26]=3;a[27]=3;a[28]=3;a[29]=3;a[30]=3; \*/

for (int k = 0; k < 6; k++) {

a[k] = "B-NM";

}

a[6] = "B-NB";

for (int l = 7; l < 16; l++) {

a[l] = "B-M ";

}

for (int l1 = 16; l1 < 21; l1++) {

a[l1] = "B-SA";

}

for (int l2 = 21; l2 < 24; l2++) {

a[l2] = "B-C ";

}

a[24] = "B-G ";

for (int l2 = 25; l2 < 28; l2++) {

a[l2] = "B-MA";

}

for (int l2 = 28; l2 < 36; l2++) {

a[l2] = "B-L ";

}

for (int l2 = 36; l2 < 40; l2++) {

a[l2] = "B-SE";

}

for (int l2 = 40; l2 < 88; l2++) {

// a[l2] = " ";

}

for (int l2 = 92; l2 < 96; l2++) {

a[l2] = "E-SE";

}

for (int l2 = 96; l2 < 99; l2++) {

a[l2] = "E-MA";

}

for (int l2 = 98; l2 < 101; l2++) {

a[l2] = "E-C ";

}

a[120]="E-C ";

for (int l2 = 101; l2 < 109; l2++) {

a[l2] = "E-L ";

}

a[109] = "E-G ";

for (int l2 = 110; l2 < 115; l2++) {

a[l2] = "E-M ";

}

for (int l2 = 128; l2 <132; l2++) {

a[l2] = "E-M ";

}

//a[125] = "E-NB";

for (int l2 = 115; l2 < 120; l2++) {

a[l2] = "E-SA";

}

// a[105] = "E-G ";

for (int l2 = 121; l2 < 127; l2++) {

a[l2] = "E-NM";

}

a[127]="E-NB";

for (int l2 = 88; l2 < 92; l2++) {

// a[l2] = " ";

}

a[46] = " W ";

a[47] = " W ";

a[48] = " W ";

a[57] = " W ";

a[58] = " W ";

a[71] = " W ";

a[72] = " W ";

a[64] = " W ";

a[75] = " W ";

for (int i = 0; i < 1; i++) {

for (int j = 0; j < 11; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(0-10)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 11; j < 22; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(11-22)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 22; j < 33; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(22-33)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 33; j < 44; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(33-43)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 44; j < 55; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(44-55)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 55; j < 66; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(55-66)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 66; j < 77; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(66-77)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 77; j < 88; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(77-88)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 88; j < 99; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(88-99)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 99; j < 110; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(99-109)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 110; j < 121; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(110-120)");

System.out.println("");

}

for (int i = 0; i < 1; i++) {

for (int j = 121; j < 132; j++) {

System.out.print("| " + a[j] + " ");

}

System.out.print("|");

System.out.print(" " + "(121-131)");

System.out.println("");

}

return a;

}

}