Tic Tac Toe:

Requirements:

Tic Tac Toe is a online or an application game.A 3\*3 matrices are required to play this game.

1. The game is played on a grid that's 3 squares by 3 squares.

2. You are X, your friend (or the computer in this case) is O. Players take turns putting their marks in empty squares.

3. The first player to get 3 of her marks in a row (up, down, across, or diagonally) is the winner.

4. When all 9 squares are full, the game is over. If no player has 3 marks in a row, the game ends in a tie.

HOW CAN I WIN AT TIC-TAC-TOE?

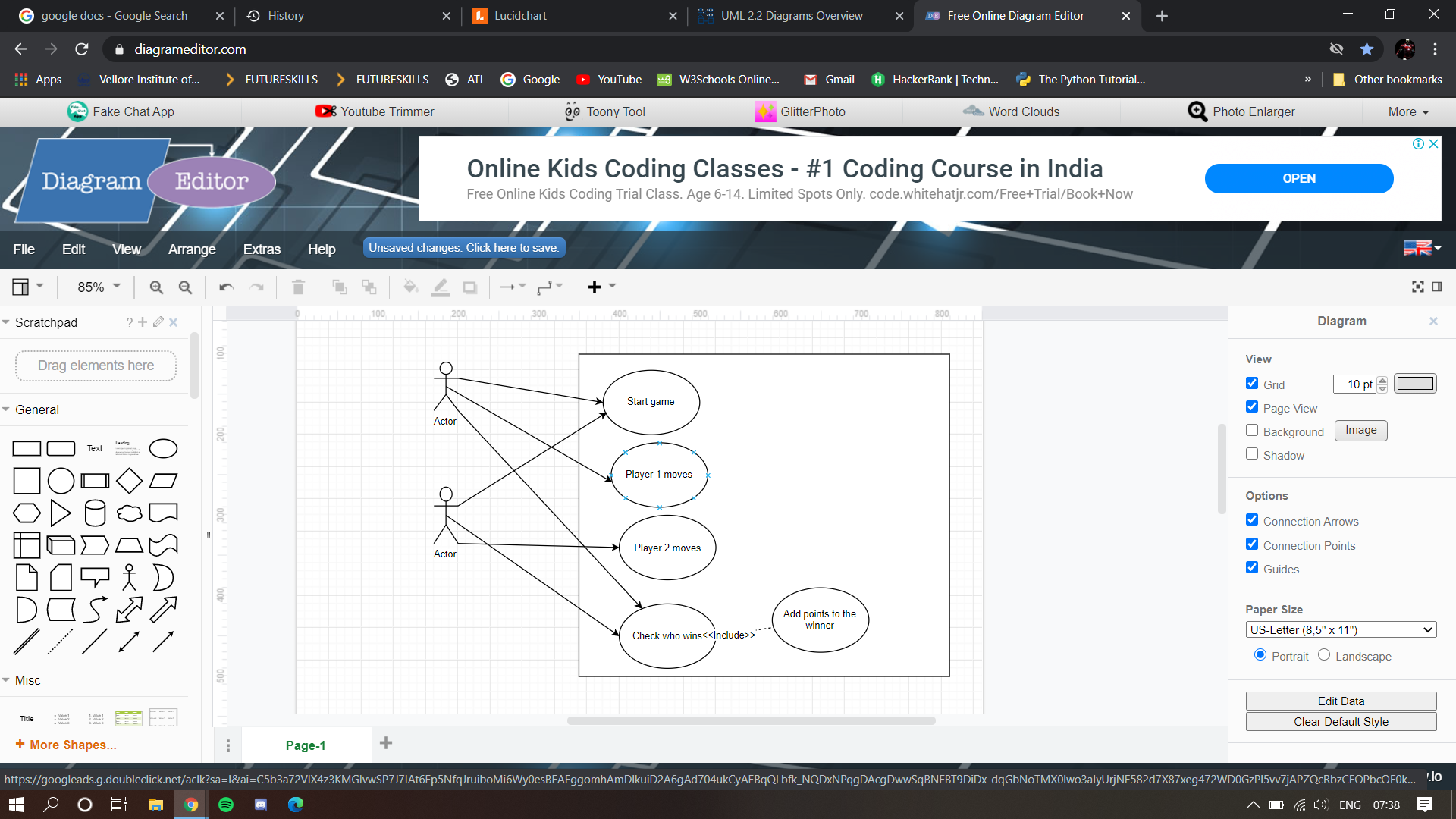
To beat the computer (or at least tie), you need to make use of a little bit of strategy. Strategy means figuring out what you need to do to win.

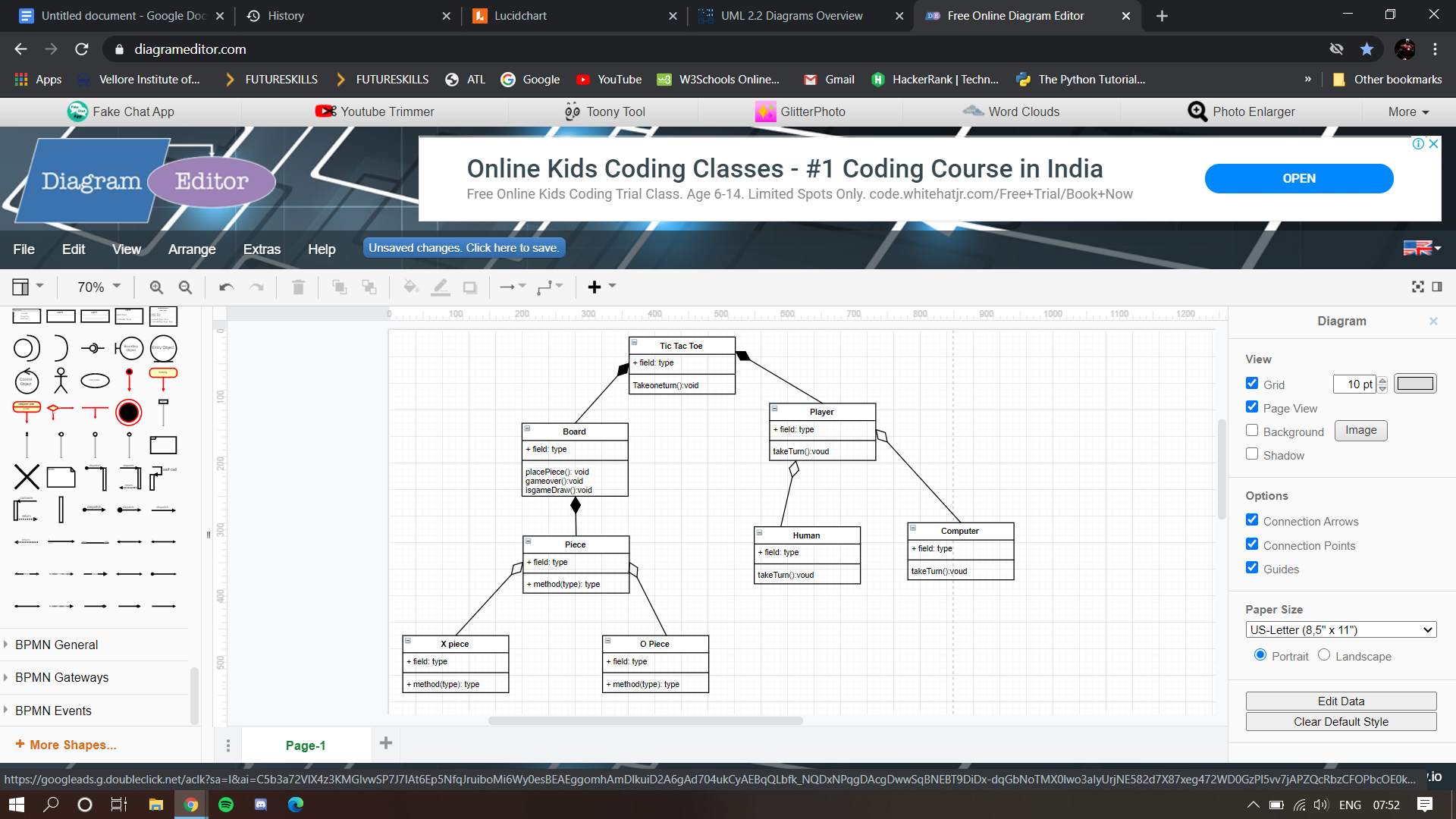
Part of your strategy is trying to figure out how to get three Xs in a row. The other part is trying to figure out how to stop the computer from getting three Os in a row.

After you put an X in a square, you start looking ahead. Where's the best place for your next X? You look at the empty squares and decide which ones are good choices—which ones might let you make three Xs in a row.

One should also have to watch where the computer puts its O. That could change what one do next. If the computer gets two Os in a row, the player has to put X in the last empty square in that row, or the computer will win. You are forced to play in a particular square or lose the game.

If you always pay attention and look ahead, you'll never lose a game of Tic-Tac-Toe. You may not win, but at least you'll tie.





#include<stdio.h>

#include<windows.h>

#include<stdlib.h>

#include<conio.h>

#include"tic.h"

struct myDataType{

int b;

char charc;

}inputValue();

int check2(char sym[9],char charc,int count){

int a;

for(a = 0;a<=6; a+=3)

if(sym[a] == charc && sym[a+1]==charc&&sym[a+2]==charc){

printf("the Winner is : %c",charc);return 1;

}

for(a = 0;a<3; a++)//it's for column

if(sym[a]==charc && sym[a+3]==charc&&sym[a+6]==charc){

printf("the Winner is : %c",charc);return 1;

}

if(sym[0]==charc && sym[4]==charc&&sym[8]==charc){

printf("the Winner is : %c",charc);return 1;

}

else if(sym[2]==charc && sym[4]==charc && sym[6]==charc){

printf("the Winner is : %c",charc);return 1;

}

else if(count==8){

printf("the Game is DRAW");

return 1;

}else return 0;

}

void run();

int check(char sym[9],char ch,int count);

struct myDataType inputValue(char sym[9],int count);

void Display(char sym[9]);

//My contribution

int check1(char sym[9],char ch,int count){

int a;

for(a = 0;a<=6; a+=3)//it's for row

if(sym[a] == ch && sym[a+1]==ch&&sym[a+2]==ch){

printf("the Winner is : %c",ch);return 1;

}

for(a = 0;a<3; a++)//it's for column

if(sym[a]==ch && sym[a+3]==ch&&sym[a+6]==ch){

printf("the Winner is : %c",ch);return 1;

}

if(sym[0]==ch && sym[4]==ch&&sym[8]==ch){

printf("the Winner is : %c",ch);return 1;

}

else if(sym[2]==ch && sym[4]==ch && sym[6]==ch){

printf("the Winner is : %c",ch);return 1;

}

else if(count==8){

printf("the Game is DRAW");

return 1;

}else return 0;

}

void run(){

int count = 0;

struct myDataType info;

char symbol[9] = {'1','2','3','4','5','6','7','8','9'};

Disp(symbol);

again:

info = inputValue(symbol,count);

symbol[info.b] = info.charc;

system("cls");

Disp(symbol);

if(check(symbol,info.charc,count)==1);

else{

count++;

goto again;

}

}

void main(){

char reStart;

again:

run();

printf("\nTo Play another match please Press 1: \nElse Any:");

scanf("%s",&reStart);

if(reStart == '1')

{

system("cls");

goto again;

}

else

exit(0);

}

int check(char sym[9],char ch,int count){

int i;

for(i = 0;i<=6; i+=3)//it's for row

if(sym[i] == ch && sym[i+1]==ch&&sym[i+2]==ch){

printf("the Winner is : %c",ch);return 1;

}

for(i = 0;i<3; i++)//it's for column

if(sym[i]==ch && sym[i+3]==ch&&sym[i+6]==ch){

printf("the Winner is : %c",ch);return 1;

}

if(sym[0]==ch && sym[4]==ch&&sym[8]==ch){

printf("the Winner is : %c",ch);return 1;

}

else if(sym[2]==ch && sym[4]==ch && sym[6]==ch){

printf("the Winner is : %c",ch);return 1;

}

else if(count==8){

printf("the Game is DRAW");

return 1;

}else return 0;

}

struct myDataType inputValue(char sym[9],int count){

char value;

int i;

struct myDataType info;

inputAgain:

if(count%2 == 0){

printf("\nEnter Your Choice X:");

}else{

printf("\nEnter Your Choice O:");

}

scanf("%s",&value);

for(i=0;i<9;i++){

if(value == sym[i]){

info.b = i;

if(count%2 == 0)

info.charc = 'X';

else

info.charc = 'O';

break;

}else{

info.b = -1;

info.charc = ' ';

}

}

if(info.b == -1){

printf("\nInput is not Valid");

goto inputAgain;

}

return info;

}

//My contribution

int checks1(char sym[9],char ch,int count){

int i;

for(i = 0;i<=6; i+=3)//it's for row

if(sym[i] == ch && sym[i+1]==ch&&sym[i+2]==ch){

printf("the Winner is : %c",ch);return 1;

}

for(i = 0;i<3; i++)//it's for column

if(sym[i]==ch && sym[i+3]==ch&&sym[i+6]==ch){

printf("the Winner is : %c",ch);return 1;

}

if(sym[0]==ch && sym[4]==ch&&sym[8]==ch){

printf("the Winner is : %c",ch);return 1;

}

else if(sym[2]==ch && sym[4]==ch && sym[6]==ch){

printf("the Winner is : %c",ch);return 1;

}

else if(count==8){

printf("the Game is DRAW");

return 1;

}else return 0;

}

int checks3(char sym[9],char ch,int count){

int i;

for(i = 0;i<=6; i+=3)//it's for row

if(sym[i] == ch && sym[i+1]==ch&&sym[i+2]==ch){

printf("the Winner is : %c",ch);return 1;

}

for(i = 0;i<3; i++)//it's for column

if(sym[i]==ch && sym[i+3]==ch&&sym[i+6]==ch){

printf("the Winner is : %c",ch);return 1;

}

if(sym[0]==ch && sym[4]==ch&&sym[8]==ch){

printf("the Winner is : %c",ch);return 1;

}

else if(sym[2]==ch && sym[4]==ch && sym[6]==ch){

printf("the Winner is : %c",ch);return 1;

}

else if(count==8){

printf("the Game is DRAW");

return 1;

}else return 0;

}

struct myDataType inputValues1(char sym[9],int count){

char value;

int i;

struct myDataType info;

inputAgain:

if(count%2 == 0){

printf("\nEnter Your Choice X:");

}else{

printf("\nEnter Your Choice O:");

}

scanf("%s",&value);

for(i=0;i<9;i++){

if(value == sym[i]){

info.b = i;

if(count%2 == 0)

info.charc = 'X';

else

info.charc = 'O';

break;

}else{

info.b = -1;

info.charc = ' ';

}

}

if(info.b == -1){

printf("\nInput is not Valid");

goto inputAgain;

}

return info;

}struct myDataType inputValues2(char sym[9],int count){

char value;

int i;

struct myDataType info;

inputAgain:

if(count%2 == 0){

printf("\nEnter Your Choice X:");

}else{

printf("\nEnter Your Choice O:");

}

scanf("%s",&value);

for(i=0;i<9;i++){

if(value == sym[i]){

info.b = i;

if(count%2 == 0)

info.charc = 'X';

else

info.charc = 'O';

break;

}else{

info.b = -1;

info.charc = ' ';

}

}

if(info.b == -1){

printf("\nInput is not Valid");

goto inputAgain;

}

return info;

}

void run1(){

int count = 0;

struct myDataType info;

char symbol[9] = {'1','2','3','4','5','6','7','8','9'};

Disp(symbol);

again:

info = inputValue(symbol,count);

symbol[info.b] = info.charc;

system("cls");

Disp(symbol);

if(check(symbol,info.charc,count)==1);

else{

count++;

goto again;

}

}

void Disp(char symbol[9]){

printf("\t\t\t\tT i c t a c t o e");

printf("\nPlayers 1 Symbol: X");

printf("\nPlayers 2 Symbol: O");

printf("\n\t\t\t | | ");

printf("\n\t\t\t %c | %c | %c ",symbol[0],symbol[1],symbol[2]);

printf("\n\t\t\t-------|-------|-------");

printf("\n\t\t\t %c | %c | %c ",symbol[3],symbol[4],symbol[5]);

printf("\n\t\t\t-------|-------|-------");

printf("\n\t\t\t %c | %c | %c ",symbol[6],symbol[7],symbol[8]);

printf("\n\t\t\t | | ");

}

Test plan

1. Introduction
2. This plan specifies the tests that Software Architects intends to execute to investigate the correctness of the TicTacToe application. The application is distributed and consists of four major threads of activity executing on as many as four different platforms.Test Items
3. The three major processes that comprise the application are the major test items. In addition, the GameBoard, which is a Java Bean, deserves special attention because of the special characteristics of the standard.Tested Features
4. All of the current features of the application will be tested. In addition, the APIs of the major classes will also be tested.Features Not Tested
5. NoneTesting Strategy and Approach
   1. Syntax
   2. DPACT abstract classes are used as the base classes for developing test classes. The methods correspond to test suites, scripts and cases.Description of Functionality
   3. At the system level, the functionality is an electronic version of the TicTacToe game.  
      Each class provides a specified functionality that is described in its documentation.Arguments for tests
   4. Mouse clicksExpected Output
   5. It is expected that the appropriate symbol will appear in the selected slot on both GameBoards. When the game completes in either the win/lost or tied states the appropriate message is printed on the GameBoard.Specific Exclusions
   6. NoneDependencies
   7. The DPACT test classes inherit from the DPACT base classes.  
      Each DPACT class uses reflection to access the internals of the objects that comprise the TicTacToe game.Test Case Success/Failure Criteria
6. At the system test level, test cases result in one of three terminal states for the game: exited, won/lost and tied.Pass/Fail Criteria for the Complete Test Cycle
7. Every primary use case must be successful for the application to pass. If any test associated with a primary use case fails, the system test fails.Entrance Criteria/Exit Criteria
8. Class testing will be conducted on a continuing basis during development. Integration testing will begin when at least one complete protocol between two classes has been developed. System testing will be initiated when developers/integration testers certify that a game has been completed.Test Suspension Criteria and Resumption Requirements
9. Tests will be sequenced to exercise progressively more of the system. Testing will be suspended when program faults prevent the tester from moving further into the system.Test Deliverables/Status Communications Vehicles
10. The DPACT-based test classes and the test report are the primary deliverables. The test report will be web-based so that developers can track progress of the testing effort and begin to repair the application before the testing process is complete.Testing Tasks
11. The primary tasks are:
    1. Test case selection
    2. Test class construction
    3. Test execution
    4. Test result evaluation
    5. Test report development
12. Hardware and Software Requirements
13. The software can be run on either unix or Windows based systems. The tests are derived from the DPACT framework.Problem Determination and Correction Responsibilities
14. The tester is responsible for completing a test report that identifies those tests that have resulted in program failure. The developer is responsible for addressing each deficiency identified in the test report.Staffing and Training Needs/Assignments
15. Fortunately all of our staff are experienced game players and need no further training to use the application. However, they do need training on calculating the possible combinations of moves.Tests Schedules
16. Testing will be completed by 6pm April 19th.Risks and Contingencies
17. There are a large number of combinations, (28\*1 = 256) of moves that can be made by the two players. There is a risk that a few of these combinations will be repeated because they are the "natural" sequences. Many combinations will not be tested at all. This risk can be mitigated by writing down all the combinations and using a specific one in each test.Approvals

This plan has been reviewed and approved by the development team leader and the QA department representative.