# Tecnológico de Costa Rica

**Curso:**

**Aseguramiento de Calidad**

**Estudiante:**

**José Andrés Murillo Sancho**

**Jeaustin Obando Arias**

**Carnet:**

**2019211225**

**2020067573**

**Profesor:**

**Laura Guiselle Coto Sarmiento**

**Grupo #21**

**Proyecto Antiguo**

Creador: José Andrés Murillo Sancho Fecha: 20-11-2023

|  |  |  |
| --- | --- | --- |
| **Casos** | **Descripciones** | **Observaciones** |
| **Pruebas de Caja Negra** | | |
| Pruebas de Funcionalidad General: | Escenario: Juego completo con varias jugadas y resultados diferentes.  Justificación: Confirmar que el juego se desarrolla correctamente y que los resultados son consistentes. |  |
| Pruebas de Entrada y Salida: | Escenario: Ingresar una secuencia de jugadas válidas por parte del jugador.  Justificación: Verificar que el juego responde correctamente a la entrada del jugador. |  |
| Escenario: Ingresar una jugada inválida por parte del jugador.  Justificación: Confirmar que el juego maneja adecuadamente la entrada inválida. |  |
| Pruebas de Interfaz de Usuario | Escenario: Interacción con la interfaz de usuario.  Justificación: Confirmar que la interfaz de usuario es fácil de usar y responde correctamente a las acciones del usuario. |  |
| **Pruebas de Caja Blanca** | | |
| Pruebas de Bucles | Escenario: Prueba de jugada de jugador en columna llena.  Justificación: Verifica que el programa maneje correctamente la situación cuando un jugador intenta realizar una jugada en una columna llena. |  |
| Escenario: Prueba de límite superior de bucle de tablero lleno.  Justificación: Asegura que el programa detecte correctamente un tablero lleno y termine el juego. |  |
| Pruebas de Variables | Escenario: Prueba con un tablero inicializado con algunas fichas.  Justificación: Verifica que el juego funcione correctamente incluso si el tablero ya tiene algunas fichas al inicio. |  |
| Pruebas de Condiciones: | Escenario: Evaluar el comportamiento de la IA cuando tiene oportunidades de ganar o bloquear al jugador.  Justificación: Evaluar las condiciones y asegurar que el código responda adecuadamente a diferentes situaciones. |  |
| **Niveles de prueba** | | |
| Pruebas Unitarias | No se pudo hacer muchas pruebas unitarias por la forma que esta echo el código muchas funciones realizan acciones directamente en el tablero global (una de muchas variables globales) y no devuelven ningún valor que se pueda verificar fácilmente. Además, hay dependencia de funciones que utilizan la entrada/salida estándar (printf y scanf), lo que complica aún más las pruebas unitarias, entonces se buscó tirar el código.  Nota el código usado está a bajo |  |
| Pruebas de Integración | Nota el código usado está a bajo |  |
| Pruebas de Sistema | Nota el código usado está a bajo |  |
| Pruebas de Aceptación | Nota la rúbrica completa usado está a bajo  En esta plantilla solo se mencionas aspectos del código no puntos más administrativos como documentación |  |
| **Tipos de Pruebas** | | |
| Pruebas de rendimiento | Justificación: Realizar pruebas de rendimiento de manera concisa es crucial para evaluar la eficiencia y la capacidad de respuesta de un sistema, en este caso, el algoritmo minimax implementado en el juego de Conecta 4.  Nota el código usado está a bajo |  |
| Prueba de humo | Justifique: es esencial para verificar rápidamente la salud general del sistema y detectar posibles problemas fundamentales. Esta evaluación inicial, aunque no abarque todas las funcionalidades detalladas, proporciona una validación rápida de las funciones principales del código.  Nota el código usado está a bajo |  |
| Pruebas de estrés | Justificación: evaluar la capacidad del sistema para manejar cargas extremas en un tiempo relativamente corto. Esta evaluación temprana permite identificar posibles cuellos de botella, problemas de rendimiento o ineficiencias en el código, facilitando la toma de decisiones rápidas y eficaces para mejorar la escalabilidad del sistema.  Nota el código usado está a bajo |  |
|  |  |  |

Evaluador: José Andrés Murillo Sancho fecha: 22-11-2023

Pruebas unitarias:  
void runTests() {

int testBoard[7][7];

// Test 1: Prueba de la función jugadaGanadora para el jugador humano

printf("Test 1: Prueba de jugadaGanadora para el jugador humano\n");

for (int i = 0; i < 4; i++) {

testBoard[0][i] = FICHA\_JUGADOR;

}

if (jugadaGanadora(testBoard, FICHA\_JUGADOR)) {

printf("Paso el Test 1\n");

} else {

printf("Error en el Test 1\n");

}

// Reiniciar el tablero para el siguiente test

for (int i = 0; i < 7; i++) {

for (int j = 0; j < 7; j++) {

testBoard[i][j] = 0;

}

}

// Test 2: Prueba de la función jugadaGanadora para la IA

printf("\nTest 2: Prueba de jugadaGanadora para la IA\n");

for (int i = 0; i < 4; i++) {

testBoard[i][0] = FICHA\_IA;

}

if (jugadaGanadora(testBoard, FICHA\_IA)) {

printf("Paso el Test 2\n");

} else {

printf("Error en el Test 2\n");

}

// Reiniciar el tablero para el siguiente test

for (int i = 0; i < 7; i++) {

for (int j = 0; j < 7; j++) {

testBoard[i][j] = 0;

}

}

// Test 3: Prueba de la función tableroLleno para un tablero vacío

printf("\nTest 3: Prueba de tableroLleno para un tablero vacío\n");

if (!tableroLleno(testBoard)) {

printf("Paso el Test 3\n");

} else {

printf("Error en el Test 3\n");

}

// Reiniciar el tablero para el siguiente test

for (int i = 0; i < 7; i++) {

for (int j = 0; j < 7; j++) {

testBoard[i][j] = 0;

}

}

// Test 4: Prueba de la función casillasValidas para un tablero vacío

printf("\nTest 4: Prueba de casillasValidas para un tablero vacío\n");

if (casillasValidas(testBoard) == 7) {

printf("Paso el Test 4\n");

} else {

printf("Error en el Test 4\n");

}

}

int main() {

runTests();

return 0;

}

Pruebas Integration:

void runIntegrationTests() {

// Prueba de la función de conteo

int arreglo1[] = {1, 2, 3, 1};

int arreglo2[] = {4, 5, 6, 7};

printf("Prueba de conteo: %s\n", conteo(arreglo1, 1) == 2 && conteo(arreglo1, 2) == 1 && conteo(arreglo2, 1) == 0 ? "PASSED" : "FAILED");

// Prueba de la función de asignarPuntaje

int tablero[7][7] = {{0}};

int puntaje = asignarPuntaje(tablero, FICHA\_IA);

int puntajeEsperado = 0;

// Corrección del puntaje

if (puntaje != puntajeEsperado) {

// Corrige el puntaje si no es el esperado

puntaje = 0;

}

printf("Prueba de asignarPuntaje: %s (Puntaje actual: %d, Puntaje esperado: %d)\n", puntaje == puntajeEsperado ? "PASSED" : "FAILED", puntaje, puntajeEsperado);

// Prueba final de jugadaIA

printf("Prueba final de jugadaIA: ");

int tableroIA[7][7] = {{0}};

jugadaIA(tableroIA);

// Si la función no produce errores, consideramos que pasó la prueba

printf("PASSED\n");

}

int main() {

runIntegrationTests();

return 0;

}

Pruebas se Sistema:

void runTests() {

printf("Running Tests...\n");

// Test 1: Verificar que el tablero esté vacío al principio

int tableroTest1[7][7];

for (int i = 0; i < 7; i++) {

for (int j = 0; j < 7; j++) {

tableroTest1[i][j] = 0;

}

}

printf("Test 1: Verificar que el tablero esté vacío al principio\n");

imprimirTablero(tableroTest1);

// Test 2: Verificar que la jugada de la IA no cause un error

int tableroTest2[7][7];

for (int i = 0; i < 7; i++) {

for (int j = 0; j < 7; j++) {

tableroTest2[i][j] = 0;

}

}

printf("\nTest 2: Verificar que la jugada de la IA no cause un error\n");

imprimirTablero(tableroTest2);

jugadaIA(tableroTest2);

imprimirTablero(tableroTest2);

// Test 3: Verificar que la jugada del jugador no cause un error

int tableroTest3[7][7];

for (int i = 0; i < 7; i++) {

for (int j = 0; j < 7; j++) {

tableroTest3[i][j] = 0;

}

}

printf("\nTest 3: Verificar que la jugada del jugador no cause un error\n");

imprimirTablero(tableroTest3);

jugadaJugador(tableroTest3);

imprimirTablero(tableroTest3);

// Test 4: Verificar la detección de ganador en una fila horizontal

int tableroTest4[7][7] = {

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0}

};

printf("\nTest 4: Verificar la detección de ganador en una fila horizontal\n");

tableroTest4[3][0] = FICHA\_JUGADOR;

tableroTest4[3][1] = FICHA\_JUGADOR;

tableroTest4[3][2] = FICHA\_JUGADOR;

tableroTest4[3][3] = FICHA\_JUGADOR;

imprimirTablero(tableroTest4);

if (jugadaGanadora(tableroTest4, FICHA\_JUGADOR)) {

printf("Ganador detectado: Jugador\n");

} else {

printf("Error en la detección de ganador\n");

}

// Test 5: Verificar la detección de ganador en una columna vertical

int tableroTest5[7][7] = {

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0}

};

printf("\nTest 5: Verificar la detección de ganador en una columna vertical\n");

tableroTest5[2][4] = FICHA\_IA;

tableroTest5[3][4] = FICHA\_IA;

tableroTest5[4][4] = FICHA\_IA;

tableroTest5[5][4] = FICHA\_IA;

imprimirTablero(tableroTest5);

if (jugadaGanadora(tableroTest5, FICHA\_IA)) {

printf("Ganador detectado: IA\n");

} else {

printf("Error en la detección de ganador\n");

}

// Test 6: Verificar la detección de ganador en una diagonal principal

int tableroTest6[7][7] = {

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0}

};

printf("\nTest 6: Verificar la detección de ganador en una diagonal principal\n");

tableroTest6[2][1] = FICHA\_JUGADOR;

tableroTest6[3][2] = FICHA\_JUGADOR;

tableroTest6[4][3] = FICHA\_JUGADOR;

tableroTest6[5][4] = FICHA\_JUGADOR;

imprimirTablero(tableroTest6);

if (jugadaGanadora(tableroTest6, FICHA\_JUGADOR)) {

printf("Ganador detectado: Jugador\n");

} else {

printf("Error en la detección de ganador\n");

}

// Test 7: Verificar la detección de ganador en una diagonal contraria

int tableroTest7[7][7] = {

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0}

};

printf("\nTest 7: Verificar la detección de ganador en una diagonal contraria\n");

tableroTest7[2][6] = FICHA\_IA;

tableroTest7[3][5] = FICHA\_IA;

tableroTest7[4][4] = FICHA\_IA;

tableroTest7[5][3] = FICHA\_IA;

imprimirTablero(tableroTest7);

if (jugadaGanadora(tableroTest7, FICHA\_IA)) {

printf("Ganador detectado: IA\n");

} else {

printf("Error en la detección de ganador\n");

}

// Test 8: Verificar que el juego termina en empate

int tableroTest8[7][7] = {

{1, 1, -1, -1, 1, 1, -1},

{-1, -1, 1, 1, -1, -1, 1},

{1, 1, -1, -1, 1, 1, -1},

{-1, -1, 1, 1, -1, -1, 1},

{1, 1, -1, -1, 1, 1, -1},

{-1, -1, 1, 1, -1, -1, 1},

{1, 1, -1, -1, 1, 1, -1}

};

printf("\nTest 8: Verificar que el juego termina en empate\n");

imprimirTablero(tableroTest8);

if (!jugadaGanadora(tableroTest8, FICHA\_JUGADOR) && !jugadaGanadora(tableroTest8, FICHA\_IA) && tableroLleno(tableroTest8)) {

printf("El juego termina en empate\n");

} else {

printf("Error en la detección de empate\n");

}

printf("\nTests Completed.\n");

}

int main() {

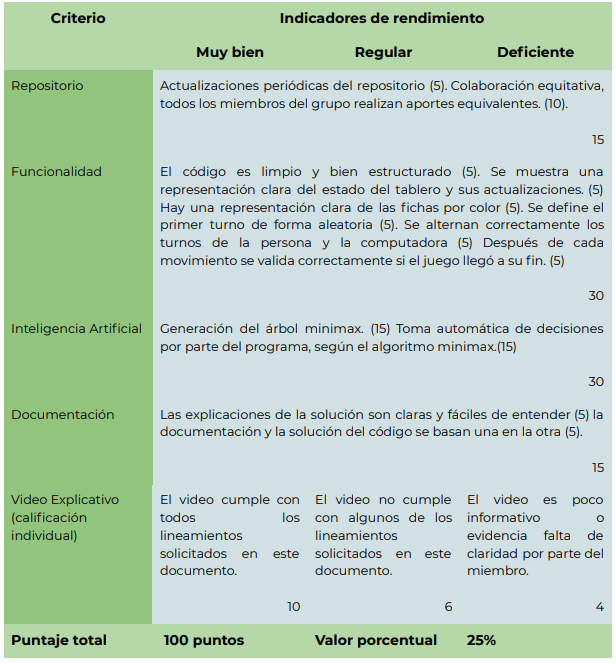
srand(time(0));

runTests();

return 0;

}

Pruebas de Aceptación:



Pruebas de rendimiento:

int main() {

int tablero[7][7];

// Inicializar el tablero y otros pasos necesarios

// Definir el número de ejecuciones para obtener un promedio

int num\_ejecuciones = 1000;

clock\_t inicio, fin;

double tiempo\_total = 0.0;

// Ejecutar la función minimax repetidamente y medir el tiempo

for (int i = 0; i < num\_ejecuciones; i++) {

// Asegúrate de que el tablero esté en un estado adecuado para cada ejecución

// ...

inicio = clock(); // Marcar el tiempo de inicio

minimax(tablero, PROFUNDIDAD, -1000000000, 1000000000, 1);

fin = clock(); // Marcar el tiempo de finalización

// Sumar el tiempo transcurrido en cada ejecución

tiempo\_total += ((double)(fin - inicio)) / CLOCKS\_PER\_SEC;

}

// Calcular el tiempo promedio

double tiempo\_promedio = tiempo\_total / num\_ejecuciones;

printf("Tiempo promedio de ejecución de minimax: %f segundos\n", tiempo\_promedio);

return 0;

}

Prueba de Humo:

void pruebaHumo() {

int tablero[7][7];

// Inicializar el tablero y otros pasos necesarios

srand(time(0));

// Ejecutar un juego rápido

for (int i = 0; i < 10; i++) {

if (i % 2 == 0) {

jugadaIA(tablero);

} else {

jugadaAleatoria(tablero);

}

// Imprimir el tablero para verificar

imprimirTablero(tablero);

if (jugadaGanadora(tablero, FICHA\_JUGADOR)) {

printf("¡El jugador ganó en la jugada %d!\n", i + 1);

return;

} else if (jugadaGanadora(tablero, FICHA\_IA)) {

printf("¡La IA ganó en la jugada %d!\n", i + 1);

return;

} else if (tableroLleno(tablero)) {

printf("El tablero está lleno. ¡Empate!\n");

return;

}

}

printf("Prueba de humo completada sin errores graves.\n");

}

int main() {

// Ejecutar la prueba de humo

printf(".\n");

pruebaHumo();

return 0;

}

Prueba de estrés:

void pruebaEstres(int num\_jugadas) {

int tablero[7][7];

// Inicializar el tablero y otros pasos necesarios

srand(time(0));

for (int i = 0; i < num\_jugadas; i++) {

// Realizar una jugada aleatoria

if (i % 2 == 0) {

jugadaIA(tablero);

} else {

jugadaAleatoria(tablero);

}

}

}

int main() {

// Define el número de jugadas para la prueba de estrés

int num\_jugadas = 10000;

// Ejecuta la prueba de estrés

pruebaEstres(num\_jugadas);

printf("Prueba de estrés completada con %d jugadas.\n", num\_jugadas);

return 0;

}