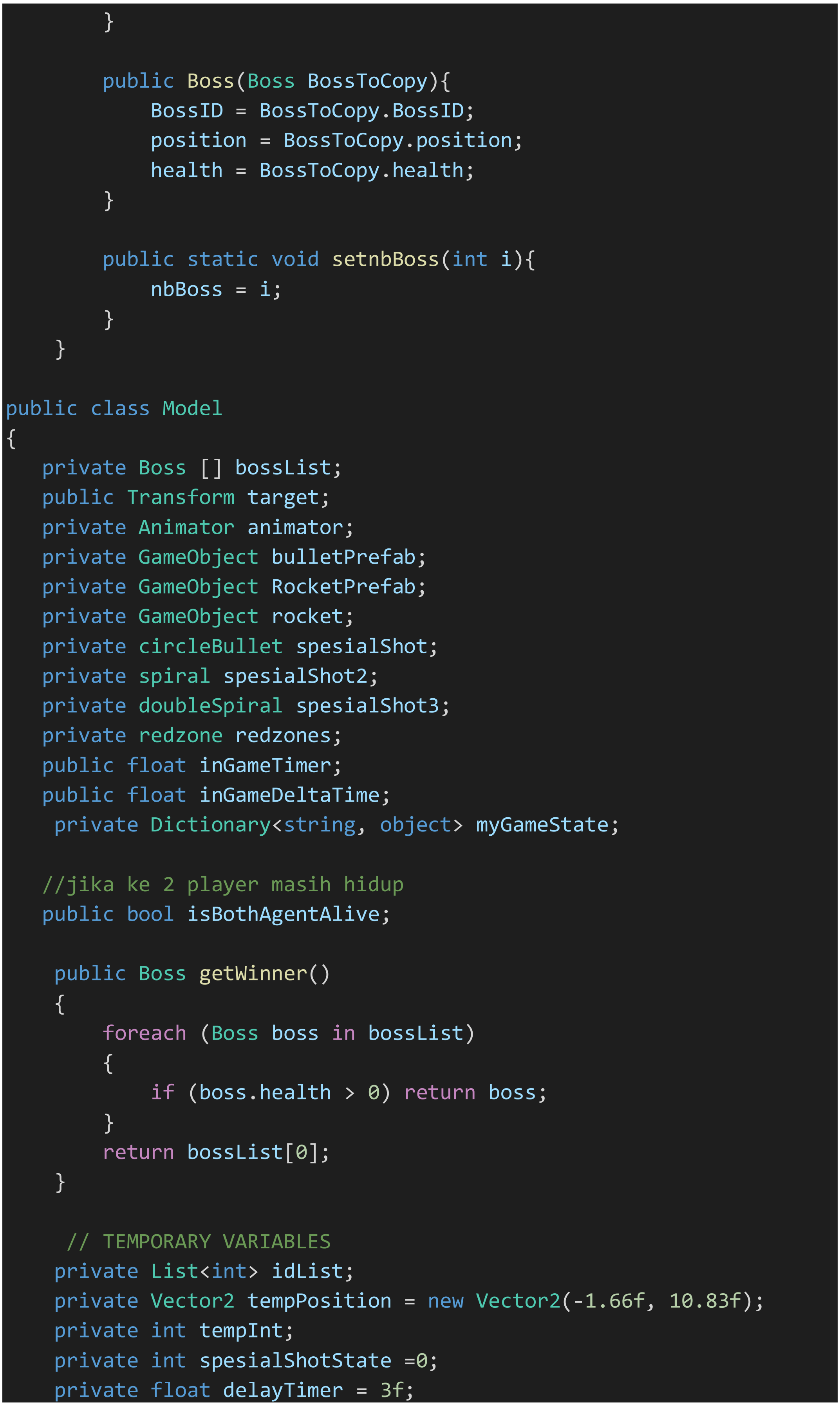
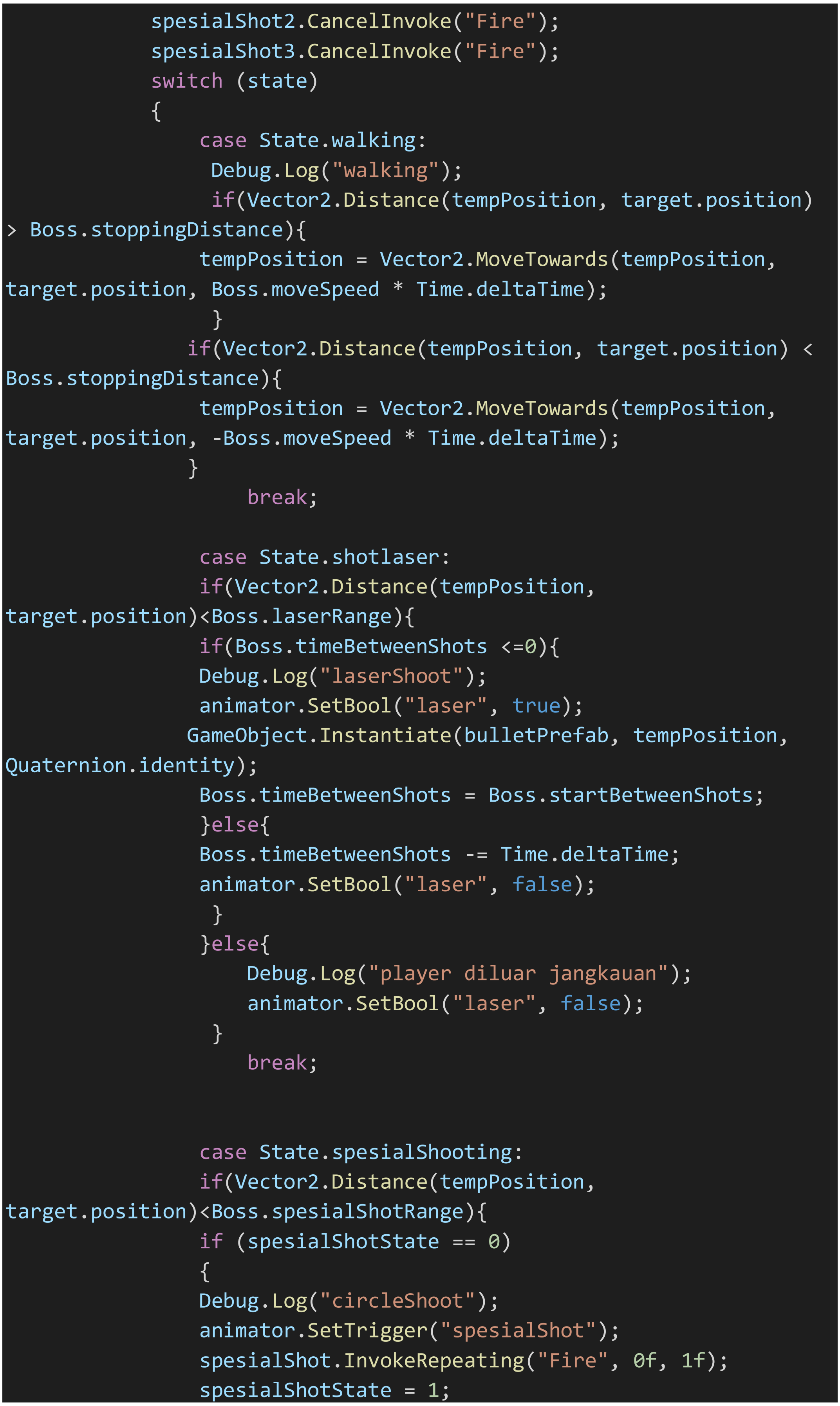
LAMPIRAN

1.1. LAMPIRAN 1. Code pada permodelan

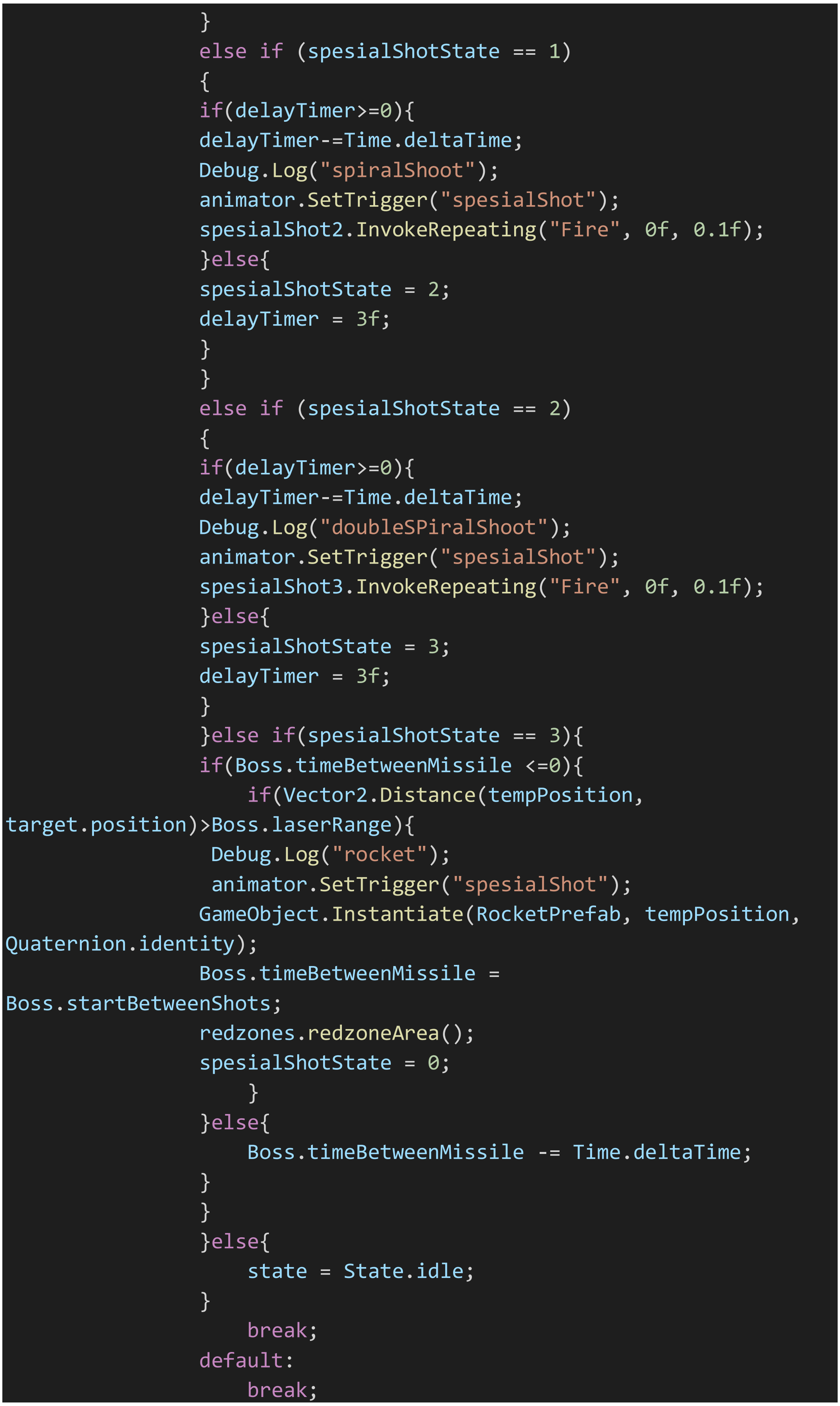
```
using System.Collections;

using System.Collections.Generic;
using UnityEngine;
using System;
using System.Linq;
using UnityEditor;
using Random = UnityEngine.Random;


namespace theLastHope{


    public enum State{
        idle,
        walking,
        shotlaser,
        spesialShooting,
        dead,
        wait,
    }

    //stats NPC Boss
    public struct Boss {
        private static int nbBoss = 0;
        public static float moveSpeed = 20f;
        public static float stoppingDistance=5f;
        public static float laserRange=6f;
        public static float spesialShotRange=11f;
        public static float startBetweenShots=0.3f;
        public static float timeBetweenShots=0f;
        public static float timeBetweenMissile=0f;


        public int BossID;
        public Vector2 position;
        public int health;


        public Boss(Vector2 pos){
            BossID = nbBoss;
            nbBoss++;
            position = pos;
            health = 750;
```



```
        }

        public Boss(Boss BossToCopy){
            BossID = BossToCopy.BossID;
            position = BossToCopy.position;
            health = BossToCopy.health;
        }

        public static void setnbBoss(int i){
            nbBoss = i;
        }
    }

public class Model
{
   private Boss [] bossList;
   public Transform target;
   private Animator animator;
   private GameObject bulletPrefab;
   private GameObject RocketPrefab;
   private GameObject rocket;
   private circleBullet spesialShot;
   private spiral spesialShot2;
   private doubleSpiral spesialShot3;
   private redzone redzones;
   public float inGameTimer;
   public float inGameDeltaTime;
    private Dictionary<string, object> myGameState;

   //jika ke 2 player masih hidup
   public bool isBothAgentAlive;

    public Boss getWinner()
    {
        foreach (Boss boss in bossList)
        {
            if (boss.health > 0) return boss;
        }
        return bossList[0];
    }

     // TEMPORARY VARIABLES
    private List<int> idList;
    private Vector2 tempPosition = new Vector2(-1.66f, 10.83f);
    private int tempInt;
    private int spesialShotState =0;
    private float delayTimer = 3f;
    public Model(Transform targetTransform, GameObject boss, int
numberOfBoss, GameObject ammo, GameObject rocket){
        Boss.setnbBoss(0);
        target = targetTransform;
        bulletPrefab = ammo;
        RocketPrefab = rocket;
        spesialShot = boss.GetComponent<circleBullet>();
        spesialShot2 = boss.GetComponent<spiral>();
        spesialShot3 = boss.GetComponent<doubleSpiral>();
        redzones = boss.GetComponent<redzone>();
        animator = boss.GetComponent<Animator>();
        myGameState = new Dictionary<string, object>();
        bossList = new Boss[numberOfBoss];
        tempPosition = new Vector2(-1.66f, 10.83f);
        isBothAgentAlive=true;

    }

    public Model(Model modelToCopy){
        inGameTimer = 0f;
        target = modelToCopy.target;
        bulletPrefab = modelToCopy.bulletPrefab;
        spesialShot = modelToCopy.spesialShot;
        spesialShot2 = modelToCopy.spesialShot2;
        spesialShot3 = modelToCopy.spesialShot3;
        RocketPrefab = modelToCopy.RocketPrefab;
        redzones = modelToCopy.redzones;
        tempPosition = modelToCopy.tempPosition;
        animator = modelToCopy.animator;
        bossList = new Boss[modelToCopy.bossList.Length];
        foreach (Boss boss in modelToCopy.bossList){
            bossList[boss.BossID] = new Boss(boss);
        }
        isBothAgentAlive = true;
        myGameState = new Dictionary<string, object>();
    }


     public void actionHandler(State state, int BossID)
    {
        if (state != State.dead)
        {

            bossList[BossID].position = tempPosition;
            spesialShot.CancelInvoke("Fire");
```



```
            spesialShot2.CancelInvoke("Fire");
            spesialShot3.CancelInvoke("Fire");
            switch (state)
            {
                case State.walking:
                 Debug.Log("walking");
                 if(Vector2.Distance(tempPosition, target.position)
> Boss.stoppingDistance){
                tempPosition = Vector2.MoveTowards(tempPosition,
target.position, Boss.moveSpeed * Time.deltaTime);
                 }
               if(Vector2.Distance(tempPosition, target.position) <
Boss.stoppingDistance){
                tempPosition = Vector2.MoveTowards(tempPosition,
target.position, -Boss.moveSpeed * Time.deltaTime);
               }
                    break;

                case State.shotlaser:
                if(Vector2.Distance(tempPosition,
target.position)<Boss.laserRange){
                if(Boss.timeBetweenShots <=0){
                Debug.Log("laserShoot");
                animator.SetBool("laser", true);
               GameObject.Instantiate(bulletPrefab, tempPosition,
Quaternion.identity);
                Boss.timeBetweenShots = Boss.startBetweenShots;
                }else{
                Boss.timeBetweenShots -= Time.deltaTime;
                animator.SetBool("laser", false);
                 }
                }else{
                    Debug.Log("player diluar jangkauan");
                    animator.SetBool("laser", false);
                 }
                    break;


                case State.spesialShooting:
                if(Vector2.Distance(tempPosition,
target.position)<Boss.spesialShotRange){
                if (spesialShotState == 0)
                {
                Debug.Log("circleShoot");
                animator.SetTrigger("spesialShot");
                spesialShot.InvokeRepeating("Fire", 0f, 1f);
                spesialShotState = 1;
```

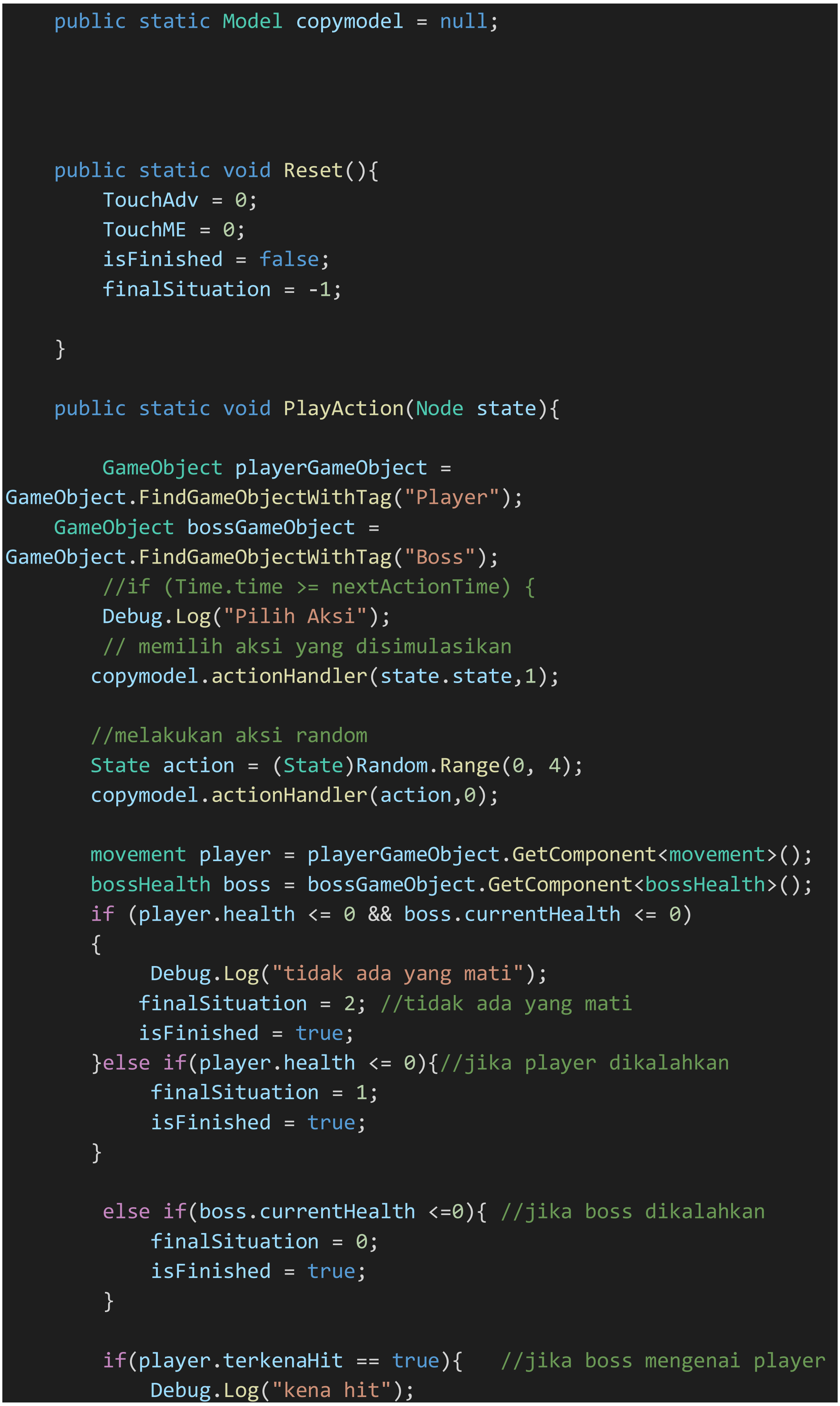


```
                }
                else if (spesialShotState == 1)
                {
                if(delayTimer>=0){
                delayTimer-=Time.deltaTime;
                Debug.Log("spiralShoot");
                animator.SetTrigger("spesialShot");
                spesialShot2.InvokeRepeating("Fire", 0f, 0.1f);
                }else{
                spesialShotState = 2;
                delayTimer = 3f;
                }
                }
                else if (spesialShotState == 2)
                {
                if(delayTimer>=0){
                delayTimer-=Time.deltaTime;
                Debug.Log("doubleSPiralShoot");
                animator.SetTrigger("spesialShot");
                spesialShot3.InvokeRepeating("Fire", 0f, 0.1f);
                }else{
                spesialShotState = 3;
                delayTimer = 3f;
                }
                }else if(spesialShotState == 3){
                if(Boss.timeBetweenMissile <=0){
                    if(Vector2.Distance(tempPosition,
target.position)>Boss.laserRange){
                 Debug.Log("rocket");
                 animator.SetTrigger("spesialShot");
                GameObject.Instantiate(RocketPrefab, tempPosition,
Quaternion.identity);
                Boss.timeBetweenMissile =
Boss.startBetweenShots;
                redzones.redzoneArea();
                spesialShotState = 0;
                    }
                }else{
                    Boss.timeBetweenMissile -= Time.deltaTime;
                }
                }
                }else{
                    state = State.idle;
                }
                    break;
                default:
                    break;
```

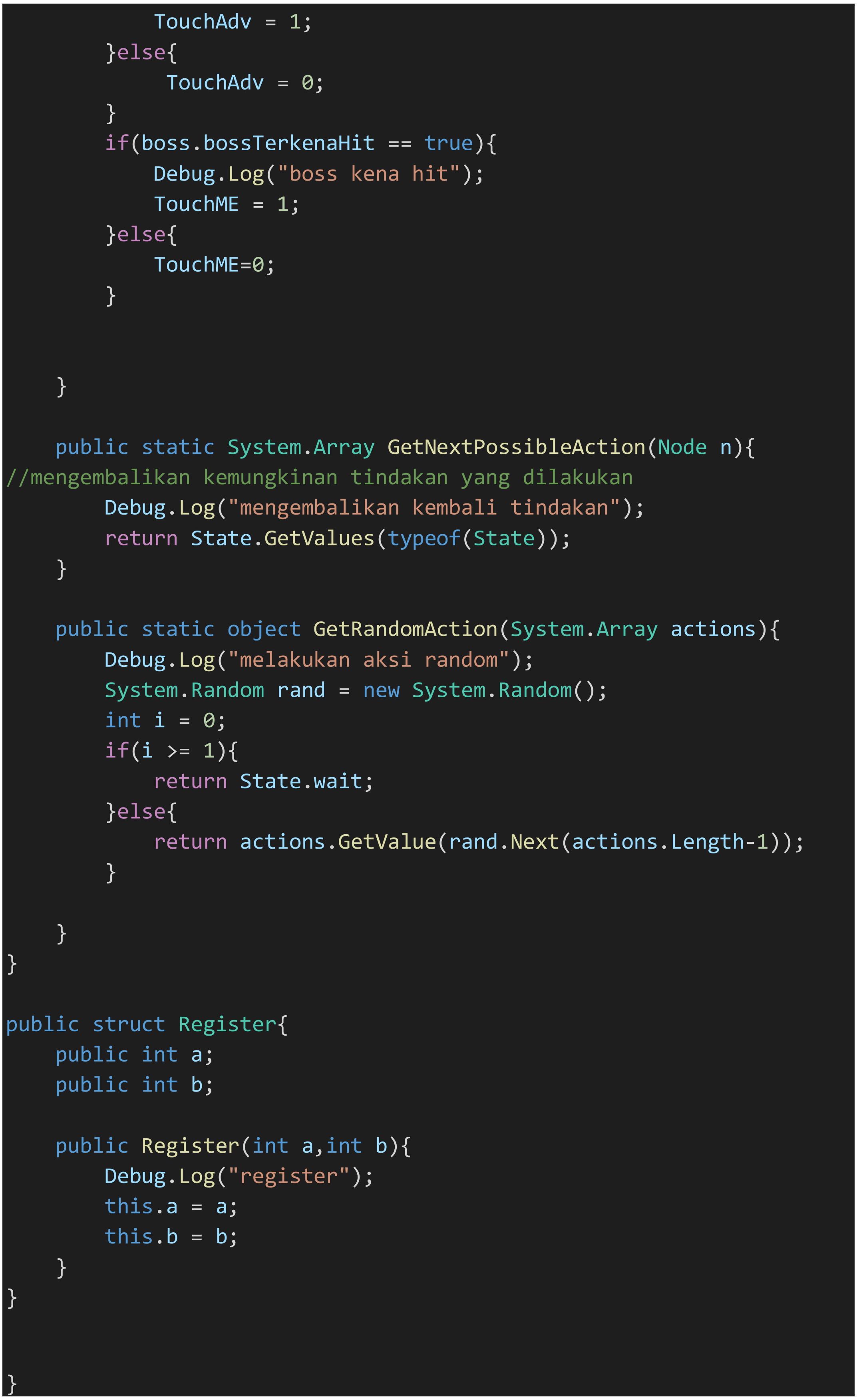
```
            }

        }
    }

     public Dictionary<string, object> getGameState()
    {
        myGameState["AgentsInfo"] = bossList;

        return myGameState;
    }


}



}
```

1.2. LAMPIRAN 2. Code pada Algoritma MCTS untuk memilih aksi

```
using System;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;
using Random = UnityEngine.Random;


namespace theLastHope{

public class GameSimul{

    public static bool isFinished = false;

    public static int finalSituation = -1; // 0 = gameover; 1 = win

    public static int TouchAdv = -1; // 0 = jika player tidak
terkena hit; 1 = jika player terkena hit

    public static int TouchME = -1; // 0 = jika npc boss tidak
terkena hit; 1 = jika npc boss terkena hit

    public static int[] ppMe = new int[4], ppAdv = new int[4];
```



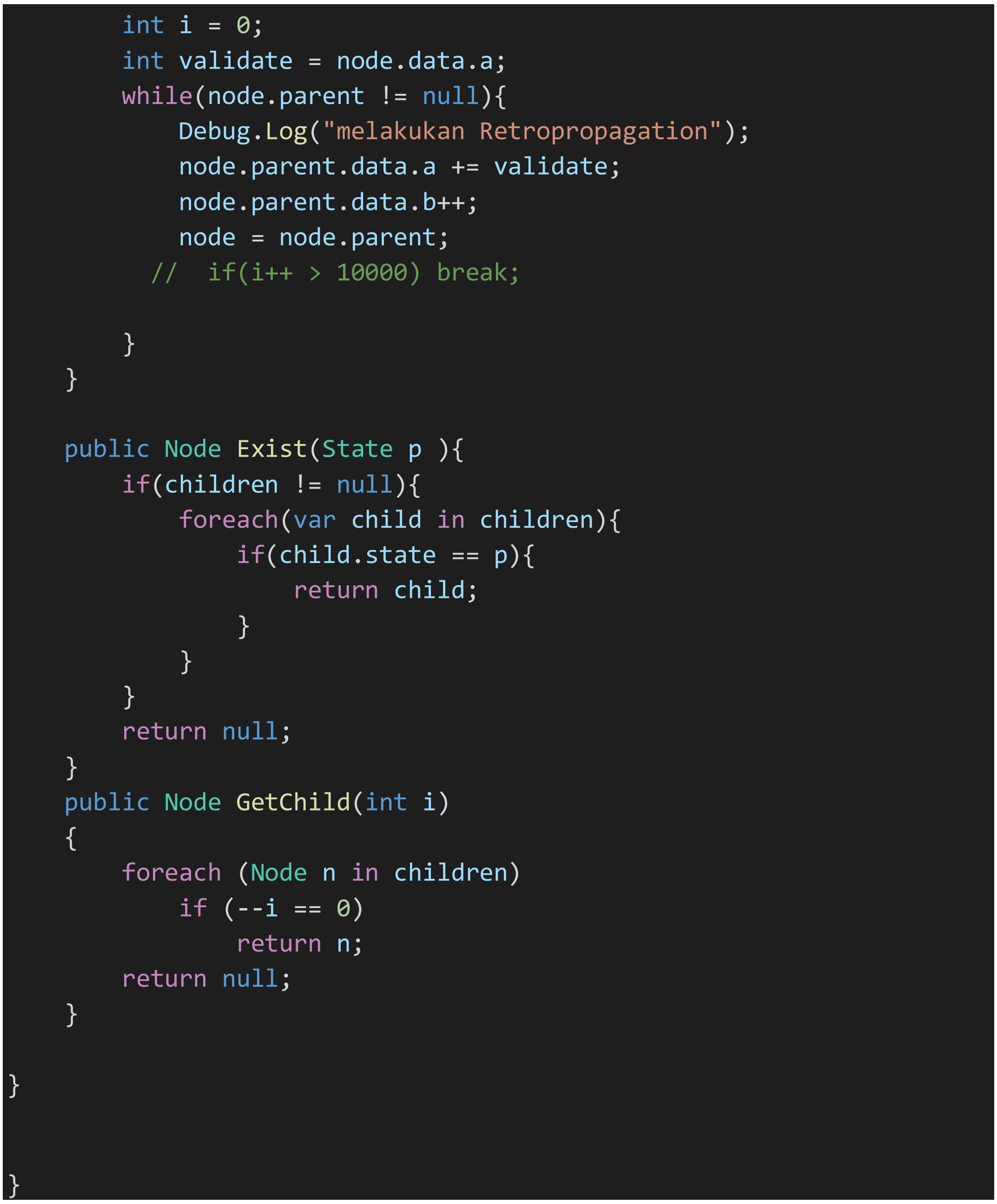
```
    public static Model copymodel = null;




    public static void Reset(){
        TouchAdv = 0;
        TouchME = 0;
        isFinished = false;
        finalSituation = -1;

    }

    public static void PlayAction(Node state){

        GameObject playerGameObject =
GameObject.FindGameObjectWithTag("Player");
    GameObject bossGameObject =
GameObject.FindGameObjectWithTag("Boss");
        //if (Time.time >= nextActionTime) {
        Debug.Log("Pilih Aksi");
        // memilih aksi yang disimulasikan
       copymodel.actionHandler(state.state,1);

       //melakukan aksi random
       State action = (State)Random.Range(0, 4);
       copymodel.actionHandler(action,0);

       movement player = playerGameObject.GetComponent<movement>();
       bossHealth boss = bossGameObject.GetComponent<bossHealth>();
       if (player.health <= 0 && boss.currentHealth <= 0)
       {
            Debug.Log("tidak ada yang mati");
           finalSituation = 2; //tidak ada yang mati
           isFinished = true;
       }else if(player.health <= 0){//jika player dikalahkan
            finalSituation = 1;
            isFinished = true;
       }

        else if(boss.currentHealth <=0){ //jika boss dikalahkan
            finalSituation = 0;
            isFinished = true;
        }

        if(player.terkenaHit == true){   //jika boss mengenai player
            Debug.Log("kena hit");
```



```
            TouchAdv = 1;
        }else{
             TouchAdv = 0;
        }
        if(boss.bossTerkenaHit == true){
            Debug.Log("boss kena hit");
            TouchME = 1;
        }else{
            TouchME=0;
        }


    }

    public static System.Array GetNextPossibleAction(Node n){
//mengembalikan kemungkinan tindakan yang dilakukan
        Debug.Log("mengembalikan kembali tindakan");
        return State.GetValues(typeof(State));
    }

    public static object GetRandomAction(System.Array actions){
        Debug.Log("melakukan aksi random");
        System.Random rand = new System.Random();
        int i = 0;
        if(i >= 1){
            return State.wait;
        }else{
            return actions.GetValue(rand.Next(actions.Length-1));
        }

    }
}

public struct Register{
    public int a;
    public int b;

    public Register(int a,int b){
        Debug.Log("register");
        this.a = a;
        this.b = b;
    }
}


}
```

1.3. LAMPIRAN 3. Code pada Node Algoritma MCTS

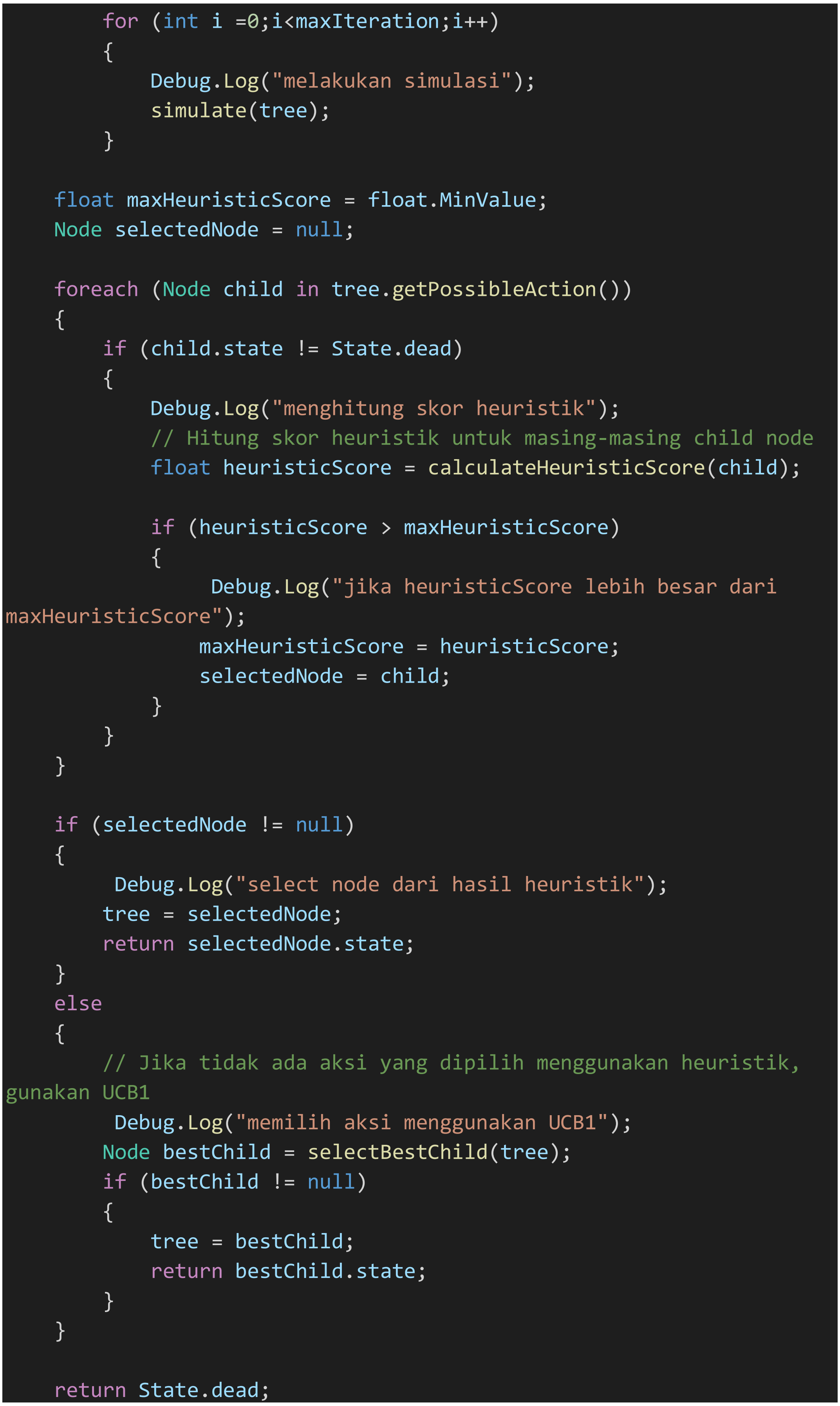
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


namespace theLastHope{

public class Node
{
    private List<Node> children;

    public Node parent;

    public State state; //state pada permodelan
    public Register data;

    public Node(Register data){
        this.data = data;
        this.children = new List<Node>();
    }
    public Node(Node parent, Register data){
        this.parent = parent;
        this.data = data;
        this.children = new List<Node>();
    }
    public Node AddChild(Register data)
    {
        Node child = new Node(data);
        children.Add(child);
        return child;
    }

    public List<Node> getPossibleAction(){ //kumpulkan node yang
sudah dibuat
        return children;
    }
    public int nbChilden(){ //mengambil sejumlah anak pada anak
pohon
        return children.Count;
    }

    public void setState(State p){ //menetapkan state
        this.state = p;
    }

    public static void Retropropagation(Node node){
```



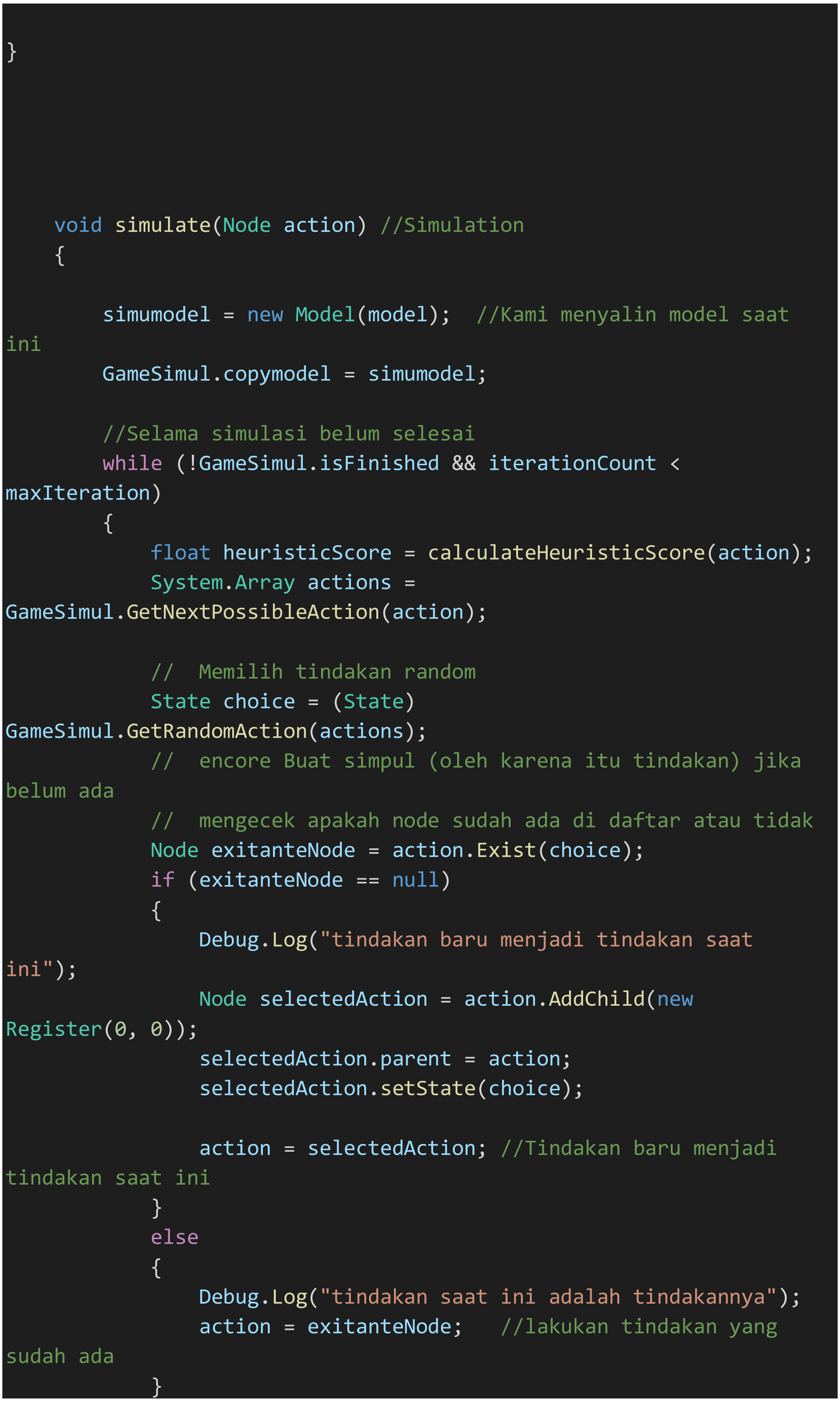
```
        int i = 0;
        int validate = node.data.a;
        while(node.parent != null){
            Debug.Log("melakukan Retropropagation");
            node.parent.data.a += validate;
            node.parent.data.b++;
            node = node.parent;
          //  if(i++ > 10000) break;

        }
    }

    public Node Exist(State p ){
        if(children != null){
            foreach(var child in children){
                if(child.state == p){
                    return child;
                }
            }
        }
        return null;
    }
    public Node GetChild(int i)
    {
        foreach (Node n in children)
            if (--i == 0)
                return n;
        return null;
    }

}


}
```

1.4. LAMPIRAN 4. Code pada Algoritma MCTS

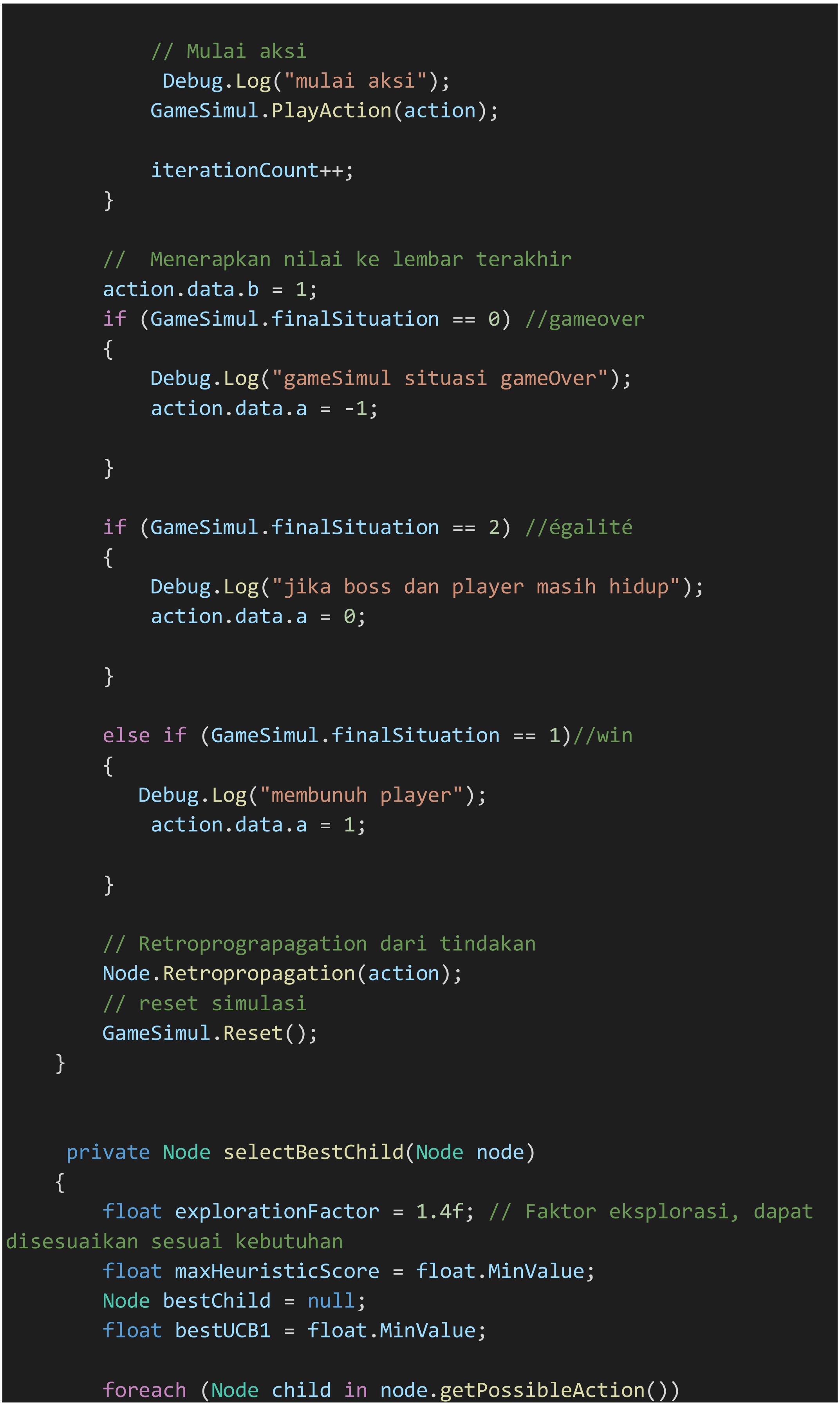
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;


namespace theLastHope{


    public class MCTS
{
    private Node tree;

    private float born;
    private Model model;
    private int BossID = 1;
    private int maxIteration = 10;
    private int iterationCount = 0;
    private Model simumodel;

     public MCTS(Model model)
    {
        tree = new Node(new Register(0, 0));
        born = 0.0f;

        this.model = model;
    }

      public bool trust()
    {
        foreach (Node n in tree.getPossibleAction())
        {
            if (n.data.b > 10)
            {
                Debug.Log("setidaknya salah satu node harus dapat
diandalkan");
                return true;
            }
        }

        return false;
    }

    public State interact() //SELECT BEST ACTION IN THREE
    {
        iterationCount = 0;
```



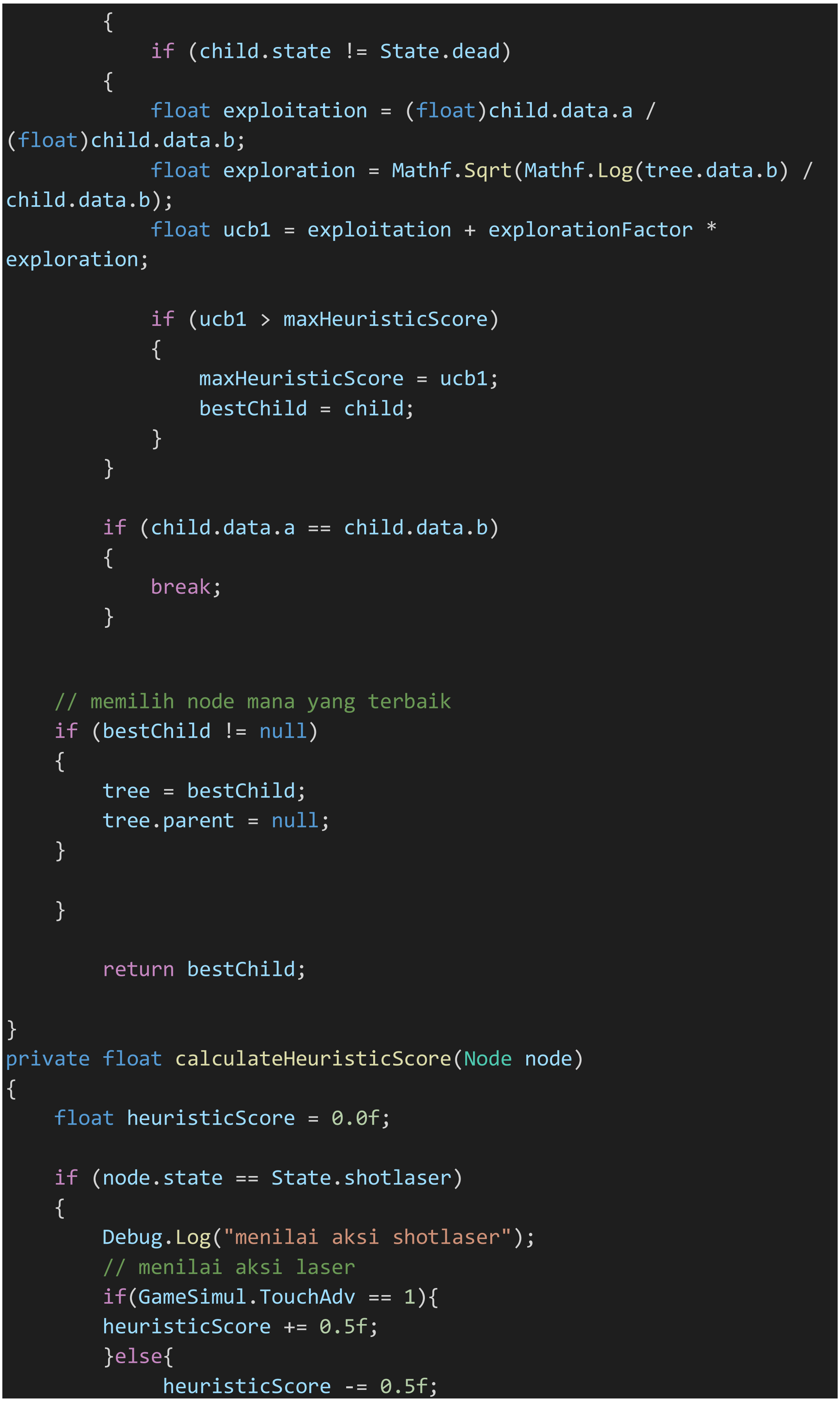
```
        for (int i =0;i<maxIteration;i++)
        {
            Debug.Log("melakukan simulasi");
            simulate(tree);
        }

    float maxHeuristicScore = float.MinValue;
    Node selectedNode = null;

    foreach (Node child in tree.getPossibleAction())
    {
        if (child.state != State.dead)
        {
            Debug.Log("menghitung skor heuristik");
            // Hitung skor heuristik untuk masing-masing child node
            float heuristicScore = calculateHeuristicScore(child);

            if (heuristicScore > maxHeuristicScore)
            {
                 Debug.Log("jika heuristicScore lebih besar dari 
maxHeuristicScore");
                maxHeuristicScore = heuristicScore;
                selectedNode = child;
            }
        }
    }

    if (selectedNode != null)
    {
         Debug.Log("select node dari hasil heuristik");
        tree = selectedNode;
        return selectedNode.state;
    }
    else
    {
        // Jika tidak ada aksi yang dipilih menggunakan heuristik, 
gunakan UCB1
         Debug.Log("memilih aksi menggunakan UCB1");
        Node bestChild = selectBestChild(tree);
        if (bestChild != null)
        {
            tree = bestChild;
            return bestChild.state;
        }
    }

    return State.dead;
```



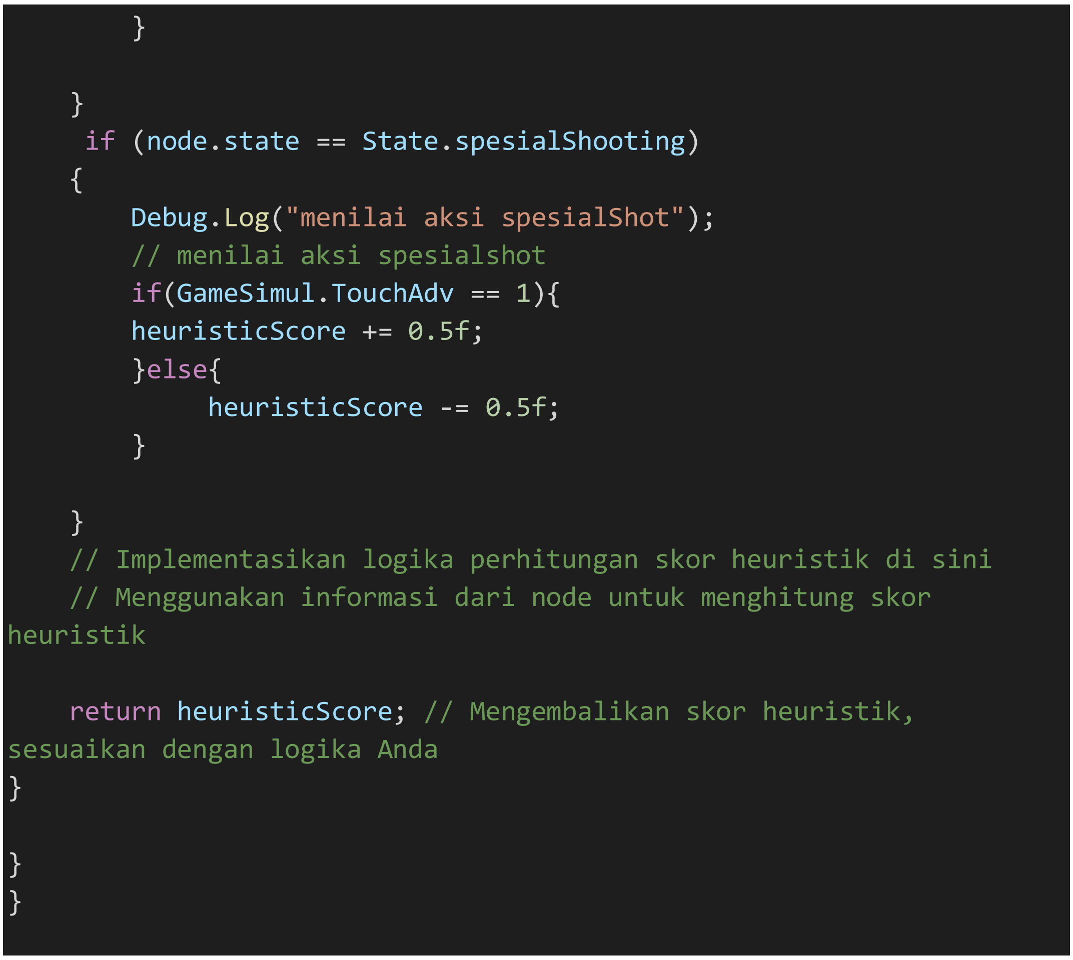
```
}




    void simulate(Node action) //Simulation
    {

        simumodel = new Model(model);  //Kami menyalin model saat
ini
        GameSimul.copymodel = simumodel;

        //Selama simulasi belum selesai
        while (!GameSimul.isFinished && iterationCount <
maxIteration)
        {
            float heuristicScore = calculateHeuristicScore(action);
            System.Array actions =
GameSimul.GetNextPossibleAction(action);

            //  Memilih tindakan random
            State choice = (State)
GameSimul.GetRandomAction(actions);
            //  encore Buat simpul (oleh karena itu tindakan) jika
belum ada
            //  mengecek apakah node sudah ada di daftar atau tidak
            Node exitanteNode = action.Exist(choice);
            if (exitanteNode == null)
            {
                Debug.Log("tindakan baru menjadi tindakan saat
ini");
                Node selectedAction = action.AddChild(new
Register(0, 0));
                selectedAction.parent = action;
                selectedAction.setState(choice);

                action = selectedAction; //Tindakan baru menjadi
tindakan saat ini
            }
            else
            {
                Debug.Log("tindakan saat ini adalah tindakannya");
                action = exitanteNode;   //lakukan tindakan yang
sudah ada
            }
```



```
            // Mulai aksi
             Debug.Log("mulai aksi");
            GameSimul.PlayAction(action);

            iterationCount++;
        }

        //  Menerapkan nilai ke lembar terakhir
        action.data.b = 1;
        if (GameSimul.finalSituation == 0) //gameover
        {
            Debug.Log("gameSimul situasi gameOver");
            action.data.a = -1;

        }

        if (GameSimul.finalSituation == 2) //égalité
        {
            Debug.Log("jika boss dan player masih hidup");
            action.data.a = 0;

        }

        else if (GameSimul.finalSituation == 1)//win
        {
           Debug.Log("membunuh player");
            action.data.a = 1;

        }

        // Retroprograpagation dari tindakan
        Node.Retropropagation(action);
        // reset simulasi
        GameSimul.Reset();
    }


     private Node selectBestChild(Node node)
    {
        float explorationFactor = 1.4f; // Faktor eksplorasi, dapat
disesuaikan sesuai kebutuhan
        float maxHeuristicScore = float.MinValue;
        Node bestChild = null;
        float bestUCB1 = float.MinValue;

        foreach (Node child in node.getPossibleAction())
```

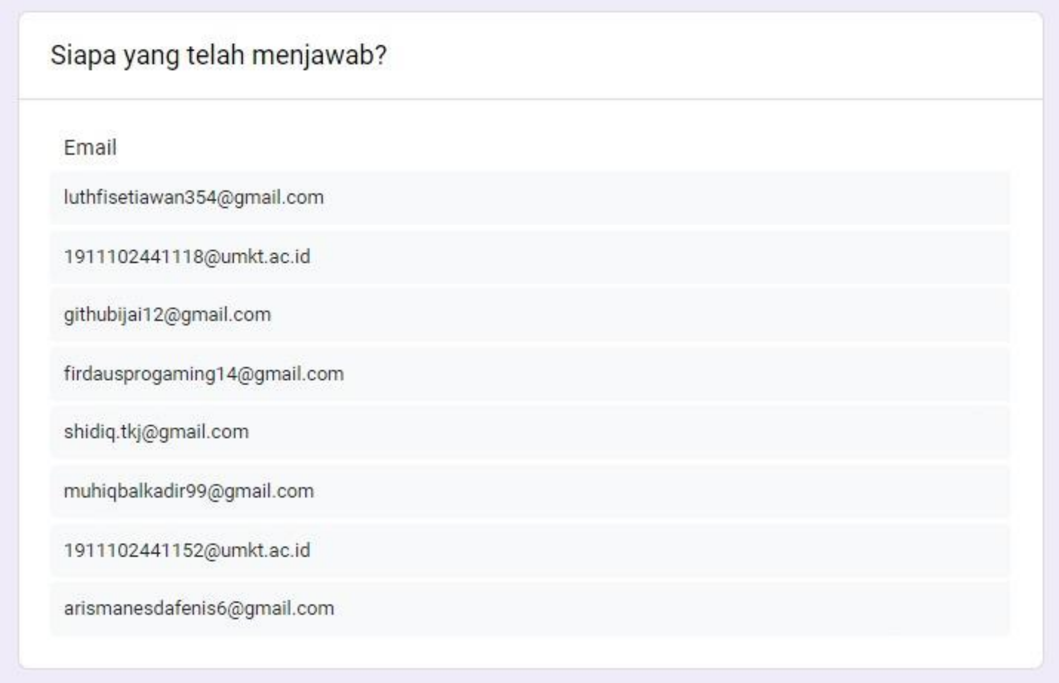


```
        {
            if (child.state != State.dead)
        {
            float exploitation = (float)child.data.a /
(float)child.data.b;
            float exploration = Mathf.Sqrt(Mathf.Log(tree.data.b) /
child.data.b);
            float ucb1 = exploitation + explorationFactor *
exploration;

            if (ucb1 > maxHeuristicScore)
            {
                maxHeuristicScore = ucb1;
                bestChild = child;
            }
        }

        if (child.data.a == child.data.b)
        {
            break;
        }


    // memilih node mana yang terbaik
    if (bestChild != null)
    {
        tree = bestChild;
        tree.parent = null;
    }

    }

        return bestChild;

}
private float calculateHeuristicScore(Node node)
{
    float heuristicScore = 0.0f;

    if (node.state == State.shotlaser)
    {
        Debug.Log("menilai aksi shotlaser");
        // menilai aksi laser
        if(GameSimul.TouchAdv == 1){
        heuristicScore += 0.5f;
        }else{
             heuristicScore -= 0.5f;
```



```
        }

    }
     if (node.state == State.spesialShooting)
    {
        Debug.Log("menilai aksi spesialShot");
        // menilai aksi spesialshot
        if(GameSimul.TouchAdv == 1){
        heuristicScore += 0.5f;
        }else{
             heuristicScore -= 0.5f;
        }

    }
    // Implementasikan logika perhitungan skor heuristik di sini
    // Menggunakan informasi dari node untuk menghitung skor
heuristik

    return heuristicScore; // Mengembalikan skor heuristik,
sesuaikan dengan logika Anda
}

}
}
```

1.5. LAMPIRAN 5. Partisipasi Dalam Uji Coba



Siapa yang telah menjawab?

Email

luthfisetiawan354@gmail.com

1911102441118@umkt.ac.id

githubijai12@gmail.com

firdausprogaming14@gmail.com

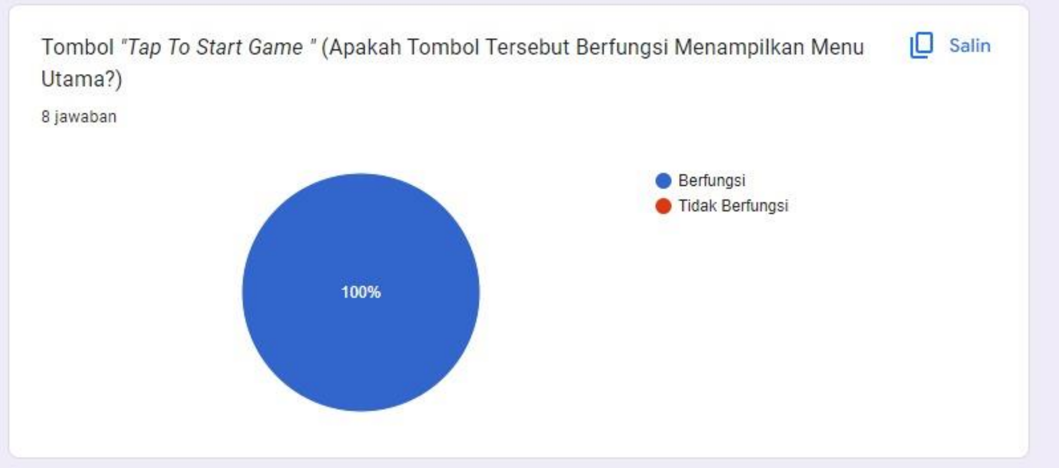
shidiq.tkj@gmail.com

muhiqbalkadir99@gmail.com

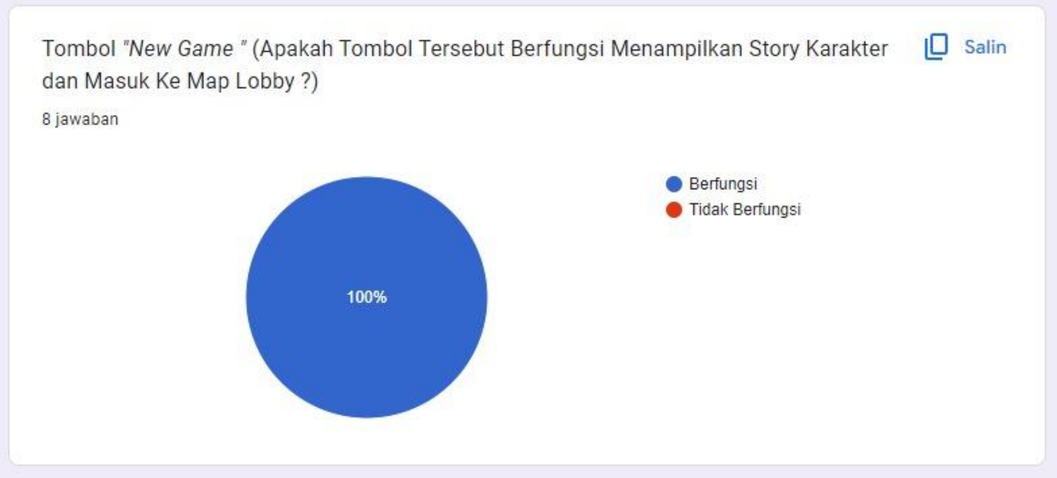
1911102441152@umkt.ac.id

arismanesdafenis6@gmail.com

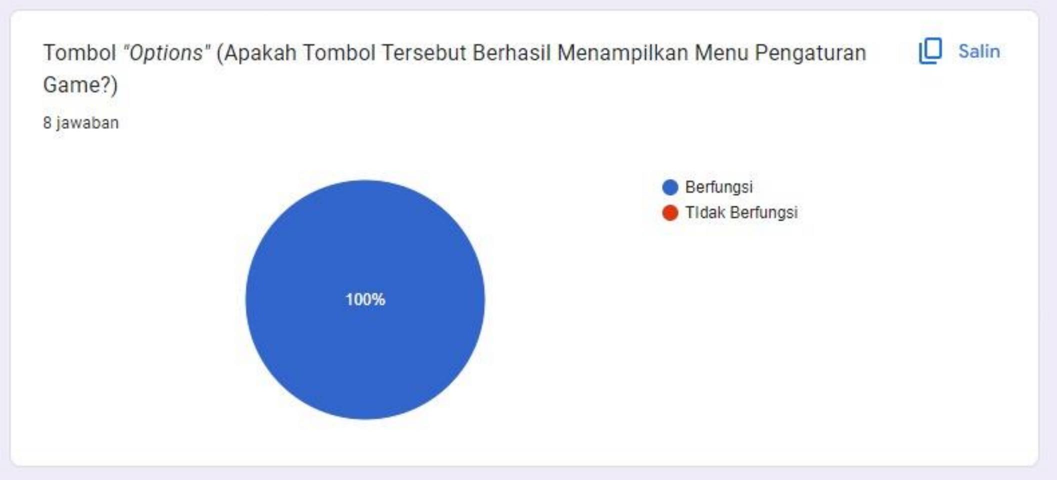
1.6. LAMPIRAN 6. Pengujian Fungsional


Tombol "*Tap To Start Game* " (Apakah Tombol Tersebut Berfungsi Menampilkan Menu Utama?)

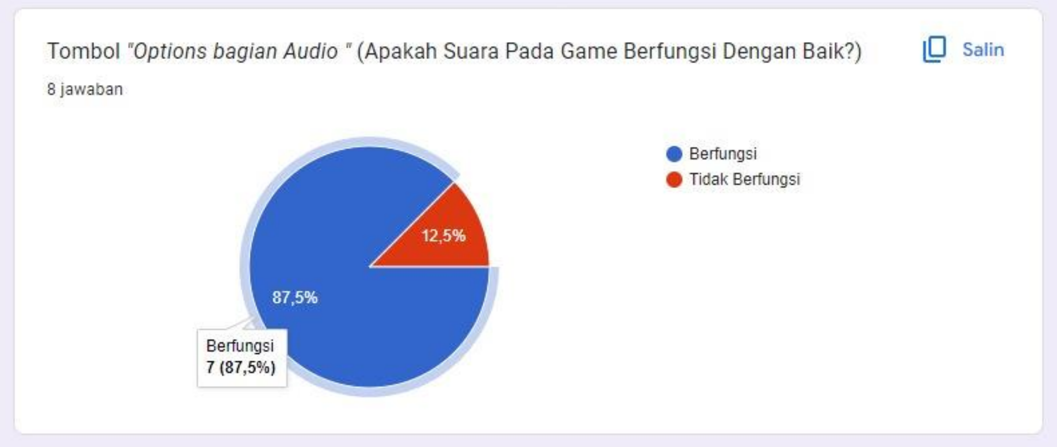
8 jawaban


Tombol "*New Game* " (Apakah Tombol Tersebut Berfungsi Menampilkan Story Karakter dan Masuk Ke Map Lobby ?)

8 jawaban

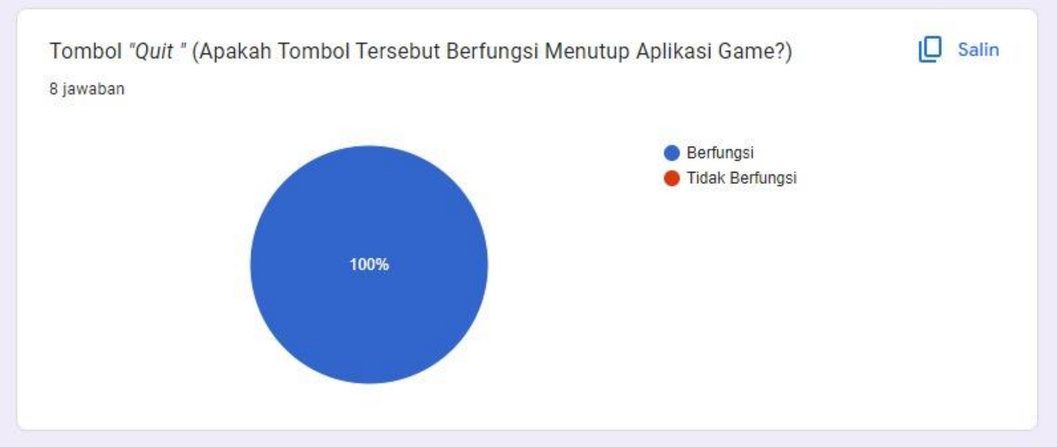

Tombol "*Options*" (Apakah Tombol Tersebut Berhasil Menampilkan Menu Pengaturan Game?)

8 jawaban



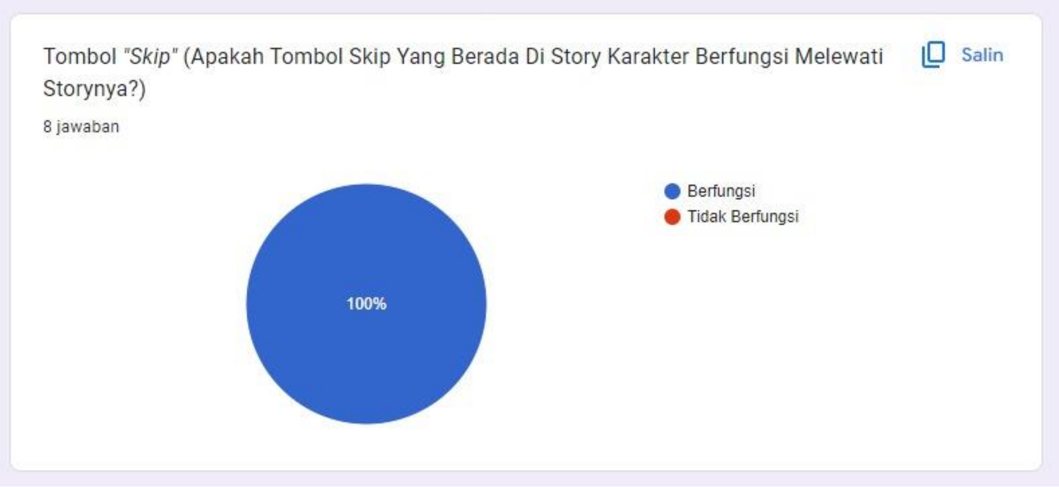
Tombol *"Options bagian Audio "* (Apakah Suara Pada Game Berfungsi Dengan Baik?)

8 jawaban



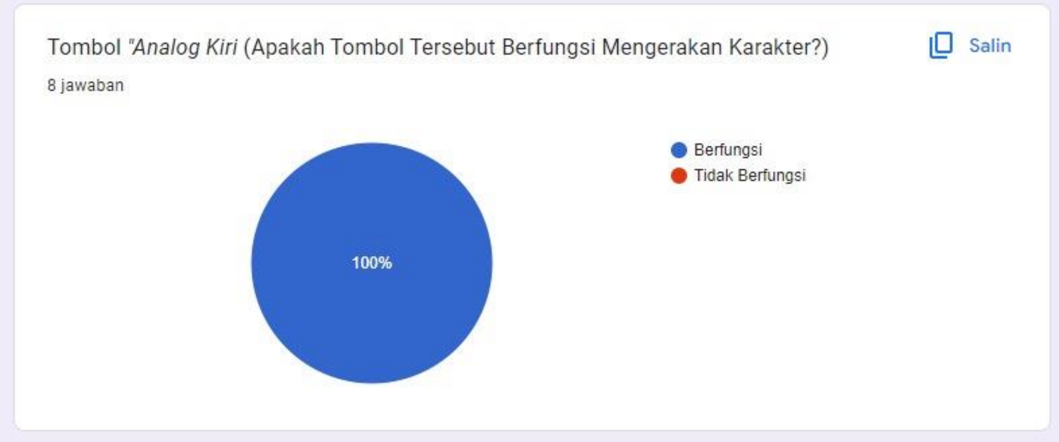
Tombol *"Quit "* (Apakah Tombol Tersebut Berfungsi Menutup Aplikasi Game?)

8 jawaban



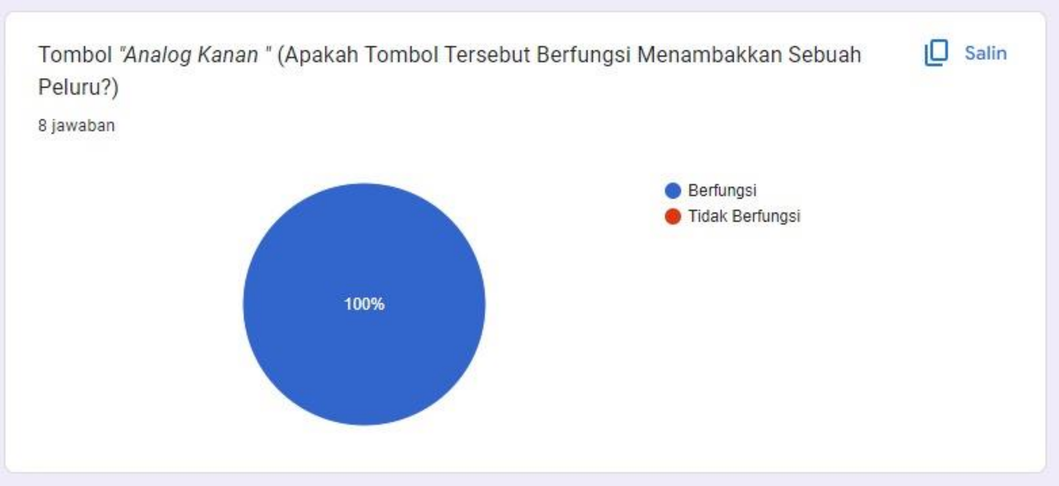
Tombol *"Skip"* (Apakah Tombol Skip Yang Berada Di Story Karakter Berfungsi Melewati Storynya?)

8 jawaban



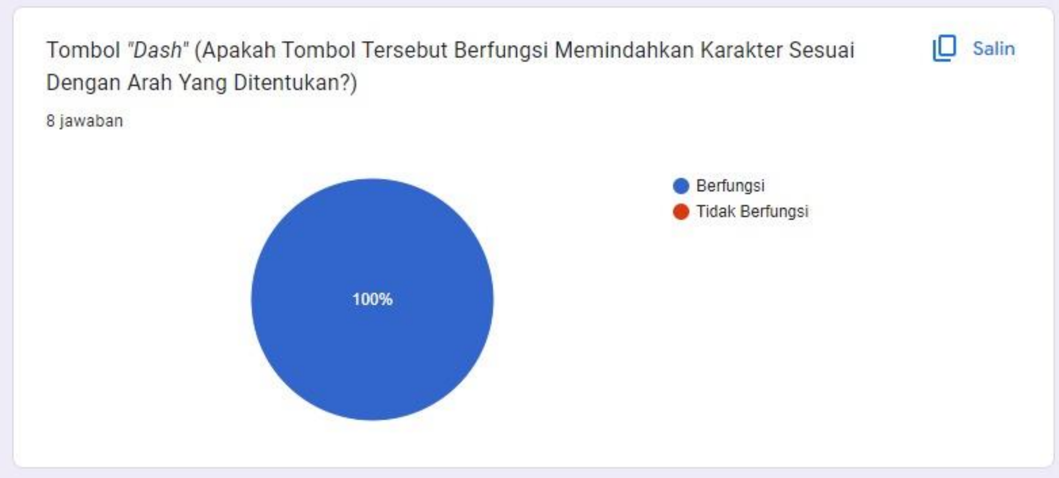
Tombol "*Analog Kiri* (Apakah Tombol Tersebut Berfungsi Mengerakan Karakter?)

8 jawaban



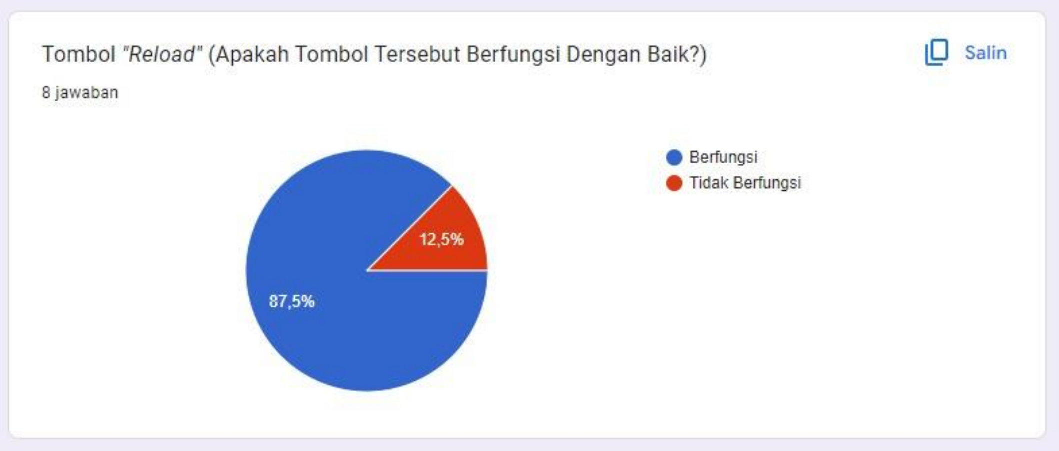
Tombol "*Analog Kanan* " (Apakah Tombol Tersebut Berfungsi Menambakkan Sebuah Peluru?)

8 jawaban



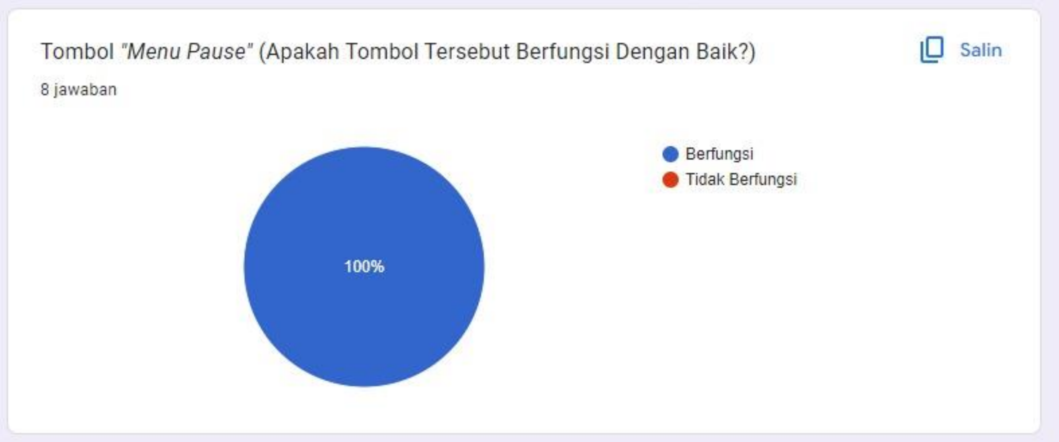
Tombol "*Dash*" (Apakah Tombol Tersebut Berfungsi Memindahkan Karakter Sesuai Dengan Arah Yang Ditentukan?)

8 jawaban



Tombol *"Reload"* (Apakah Tombol Tersebut Berfungsi Dengan Baik?)

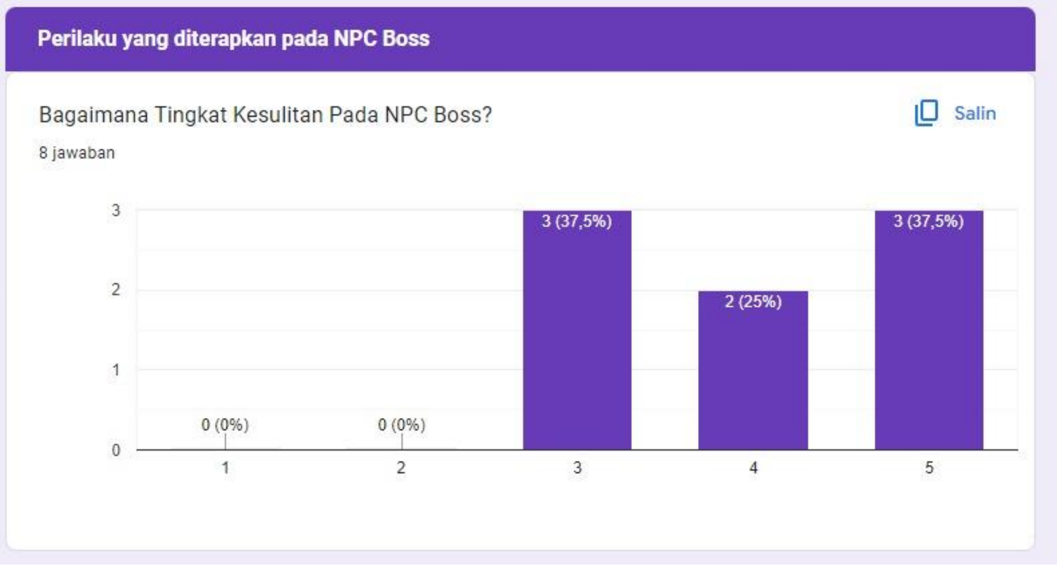
8 jawaban



Tombol *"Menu Pause"* (Apakah Tombol Tersebut Berfungsi Dengan Baik?)

8 jawaban

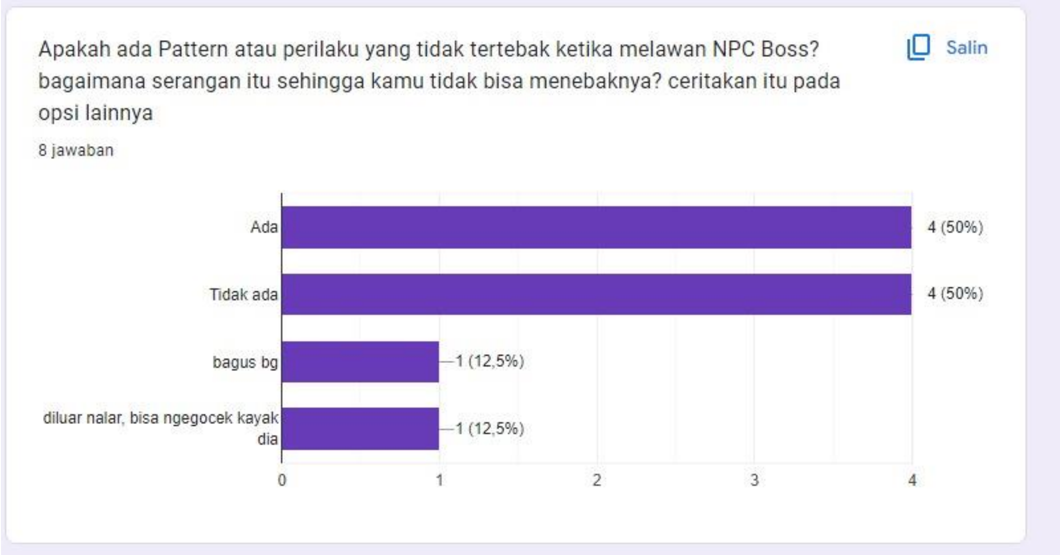
1.7. LAMPIRAN 7. Pengujian BlackBox



Perilaku yang diterapkan pada NPC Boss

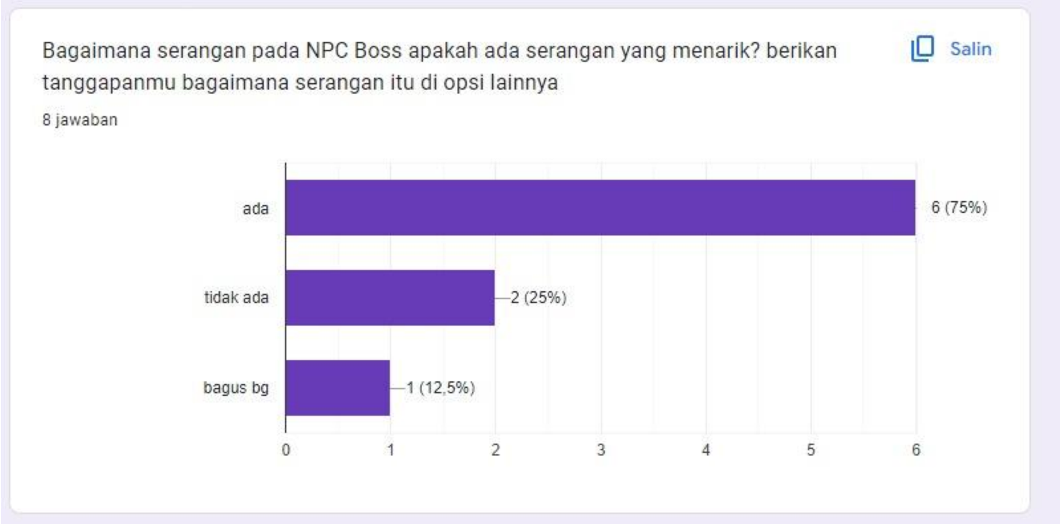
Bagaimana Tingkat Kesulitan Pada NPC Boss?

8 jawaban



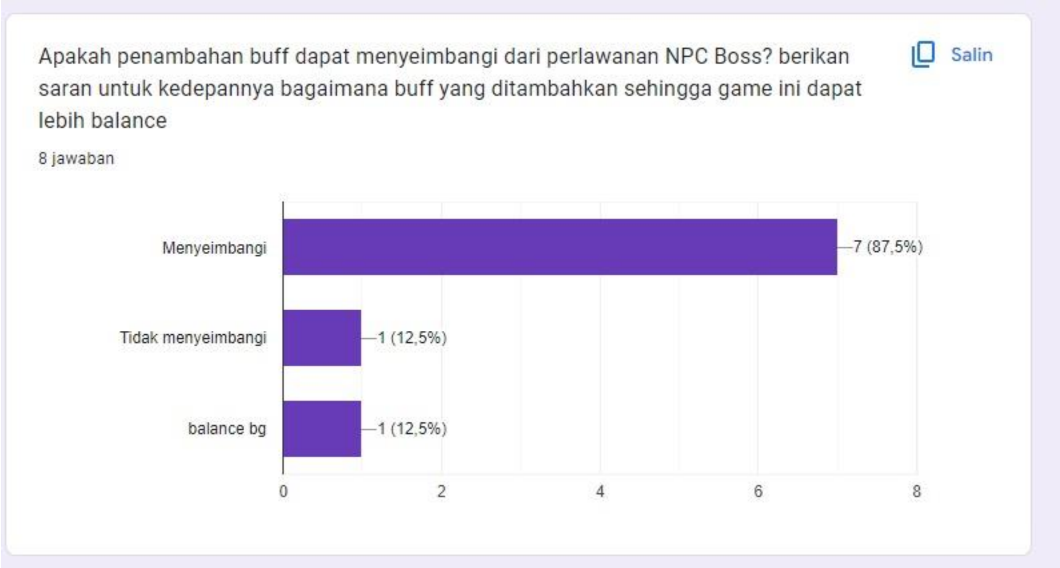
Apakah ada Pattern atau perilaku yang tidak tertebak ketika melawan NPC Boss? bagaimana serangan itu sehingga kamu tidak bisa menebaknya? ceritakan itu pada opsi lainnya

8 jawaban



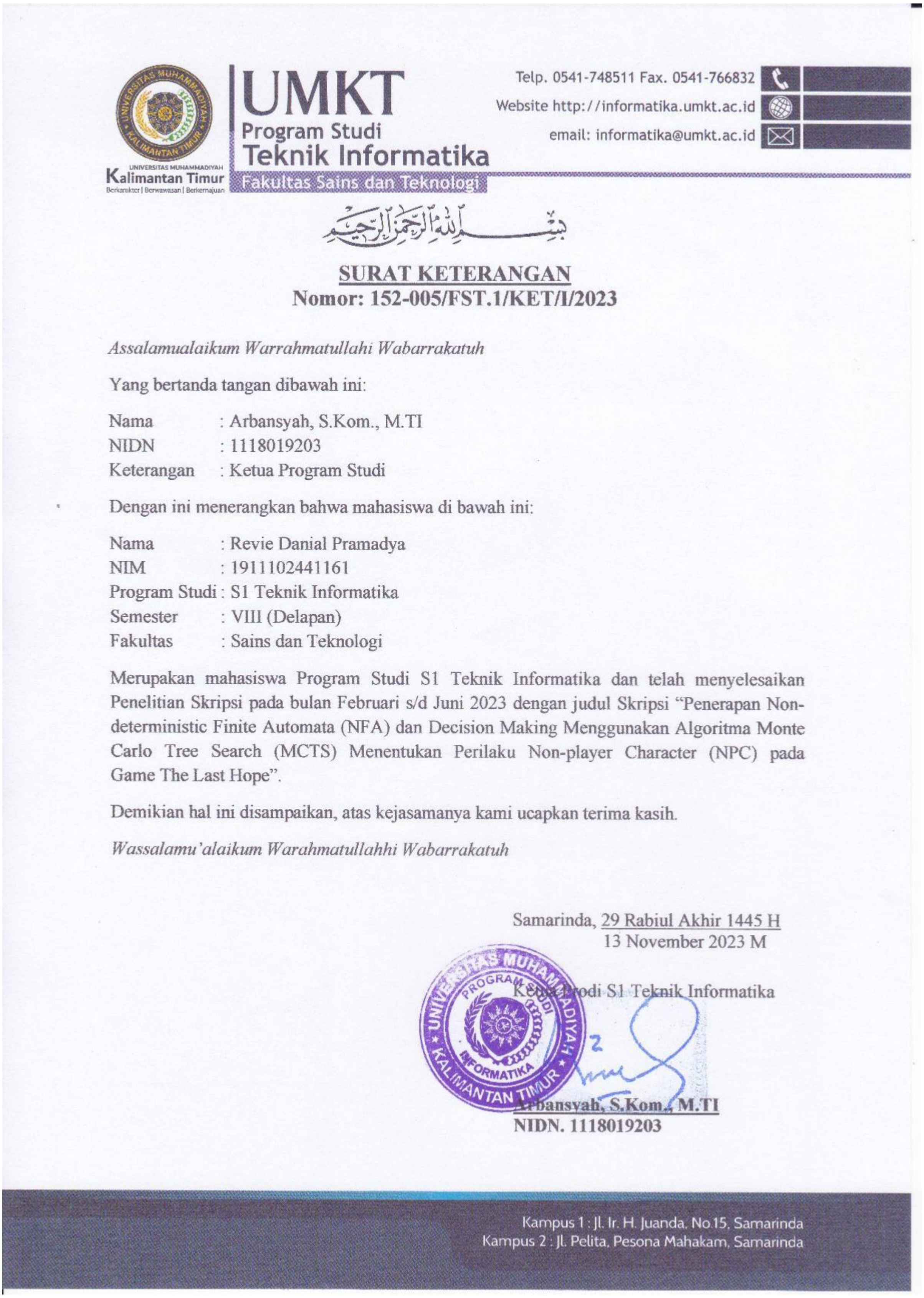
Bagaimana serangan pada NPC Boss apakah ada serangan yang menarik? berikan tanggapanmu bagaimana serangan itu di opsi lainnya

8 jawaban



Apakah penambahan buff dapat menyeimbangi dari perlawanan NPC Boss? berikan saran untuk kedepannya bagaimana buff yang ditambahkan sehingga game ini dapat lebih balance

8 jawaban



UNIVERSITAS MUHAMMADIYAH
Kalimantan Timur
Berkarakter | Berwawasan | Berkemajuan

UMKT
Program Studi
Teknik Informatika
Fakultas Sains dan Teknologi

Telp. 0541-748511 Fax. 0541-766832
Website http://informatika.umkt.ac.id
email: informatika@umkt.ac.id

بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

SURAT KETERANGAN
Nomor: 152-005/FST.1/KET/I/2023

Assalamualaikum Warrahmatullahi Wabarrakatuh

Yang bertanda tangan dibawah ini:

Nama	: Arbansyah, S.Kom., M.TI
NIDN	: 1118019203
Keterangan	: Ketua Program Studi

Dengan ini menerangkan bahwa mahasiswa di bawah ini:

Nama	: Revie Danial Pramadya
NIM	: 1911102441161
Program Studi	: S1 Teknik Informatika
Semester	: VIII (Delapan)
Fakultas	: Sains dan Teknologi

Merupakan mahasiswa Program Studi S1 Teknik Informatika dan telah menyelesaikan Penelitian Skripsi pada bulan Februari s/d Juni 2023 dengan judul Skripsi "Penerapan Non-deterministic Finite Automata (NFA) dan Decision Making Menggunakan Algoritma Monte Carlo Tree Search (MCTS) Menentukan Perilaku Non-player Character (NPC) pada Game The Last Hope".

Demikian hal ini disampaikan, atas kejasamanya kami ucapkan terima kasih.

Wassalamu'alaikum Warahmatullahhi Wabarrakatuh

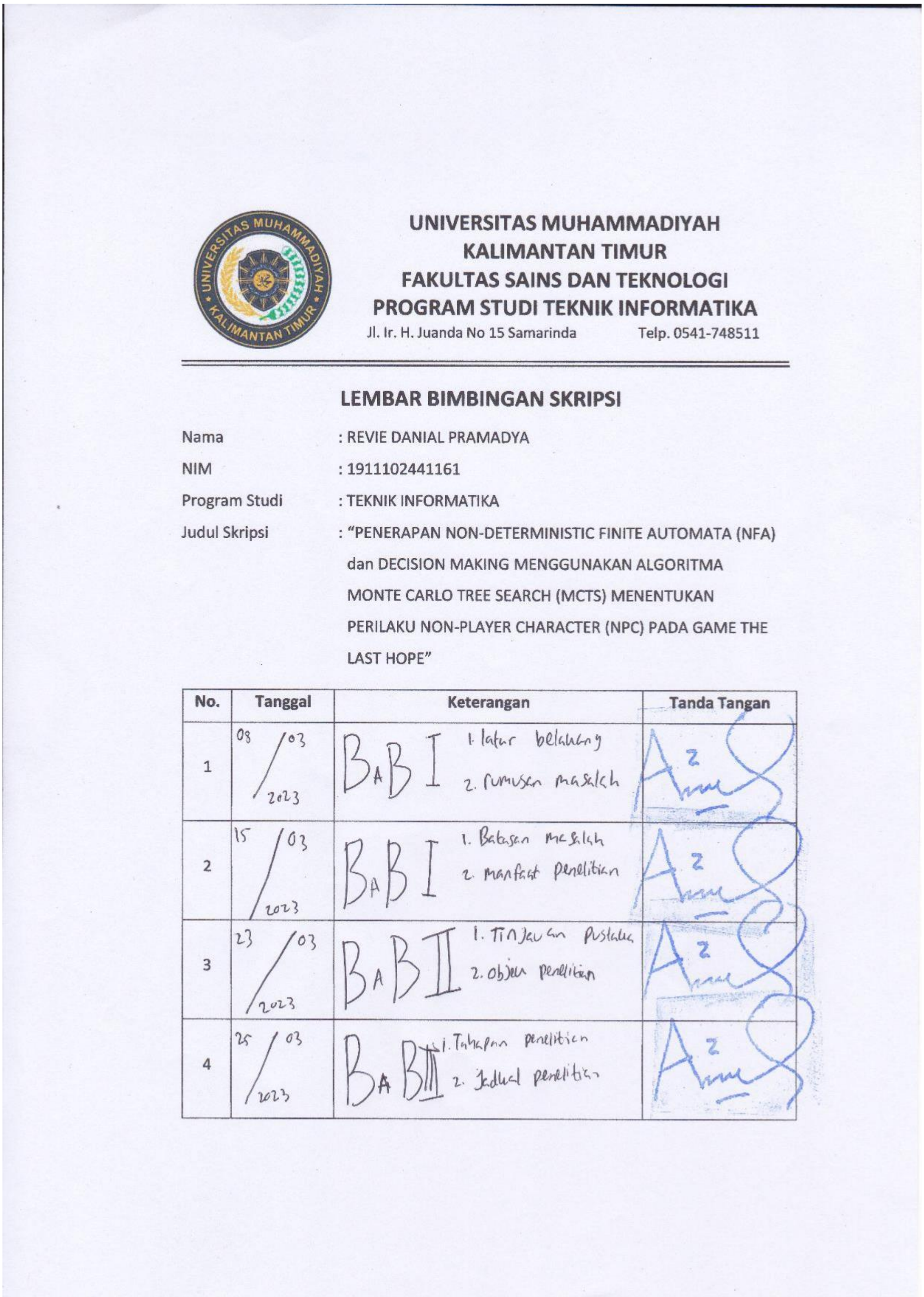
Samarinda, 29 Rabiul Akhir 1445 H
13 November 2023 M

Ketua Prodi S1 Teknik Informatika

UNIVERSITAS MUHAMMADIYAH KALIMANTAN TIMUR
PROGRAM STUDI INFORMATIKA

Arbansyah, S.Kom., M.TI
NIDN. 1118019203

Kampus 1 : Jl. Ir. H. Juanda, No.15, Samarinda
Kampus 2 : Jl. Pelita, Pesona Mahakam, Samarinda



UNIVERSITAS MUHAMMADIYAH
KALIMANTAN TIMUR
FAKULTAS SAINS DAN TEKNOLOGI
PROGRAM STUDI TEKNIK INFORMATIKA
Jl. Ir. H. Juanda No 15 Samarinda Telp. 0541-748511

LEMBAR BIMBINGAN SKRIPSI

Nama : REVIE DANIAL PRAMADYA

NIM : 1911102441161

Program Studi : TEKNIK INFORMATIKA

Judul Skripsi : "PENERAPAN NON-DETERMINISTIC FINITE AUTOMATA (NFA) dan DECISION MAKING MENGGUNAKAN ALGORITMA MONTE CARLO TREE SEARCH (MCTS) MENENTUKAN PERILAKU NON-PLAYER CHARACTER (NPC) PADA GAME THE LAST HOPE"

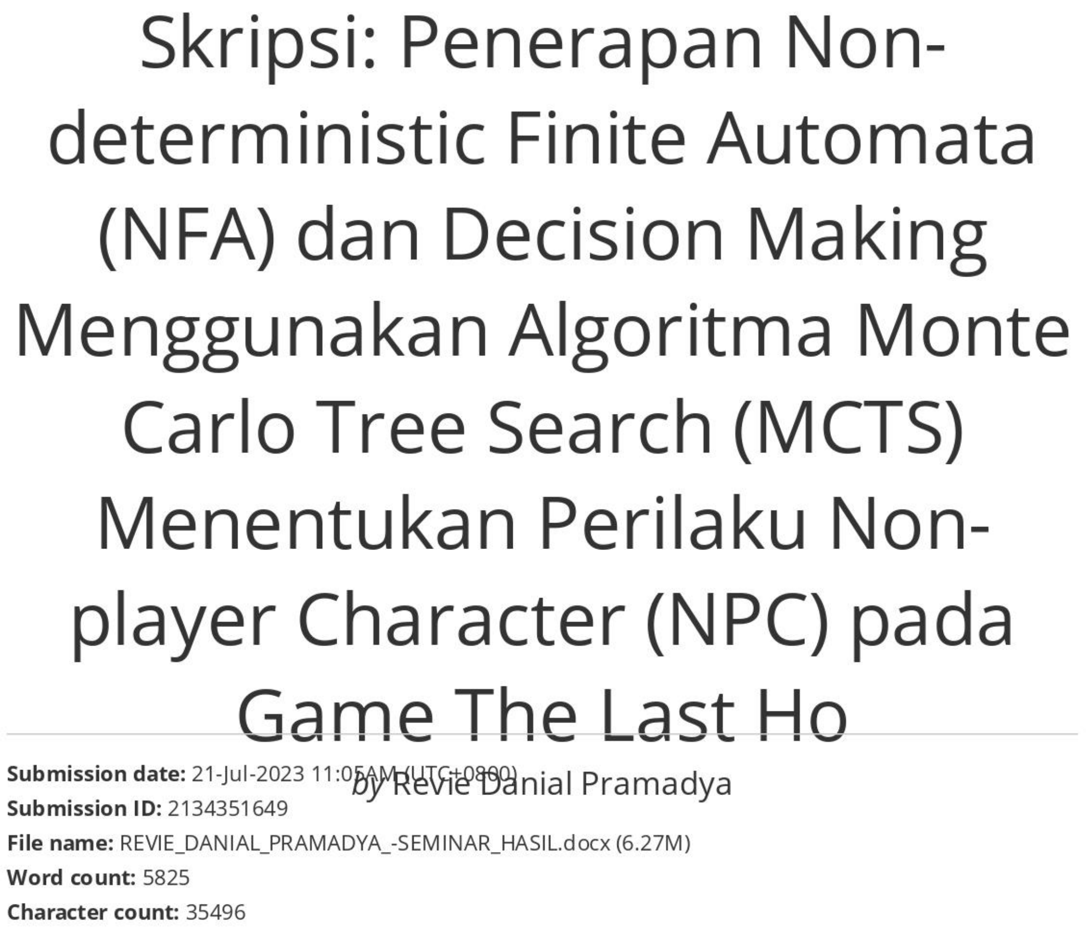
No.	Tanggal	Keterangan	Tanda Tangan
1	08/03/2023	BAB I 1. latar belakang 2. rumusan masalah	
2	15/03/2023	BAB I 1. Batasan masalah 2. manfaat penelitian	
3	23/03/2023	BAB II 1. Tinjauan pustaka 2. objek penelitian	
4	25/03/2023	BAB III 1. Tahapan penelitian 2. Jadwal penelitian	

5	10/05/2023	BAB IV	1. pembahasan tahapan penelitian	
6	15/05/2023	BAB IV	1. membahas konsep algortma 2. menentukan gameplay	
7	22/06/2023	BAB IV	1. membahas black box 2. membahas fungsional	
8	10/07/2023	BAB IV	1. Revisi komplekitas algoritma	
9	18/07/2023	Jurnal	1. Penulisan 2. tataletak	
10	20/07/2023	Jurnal & BAB V	1. Revisi kesimpulan	

Samarinda, 10 Juli 2023

Dosen Pembimbing

Arbansyah, S.Kom, M.TI



Skripsi: Penerapan Non-deterministic Finite Automata (NFA) dan Decision Making Menggunakan Algoritma Monte Carlo Tree Search (MCTS) Menentukan Perilaku Non-player Character (NPC) pada Game The Last Ho

by Revie Danial Pramadya

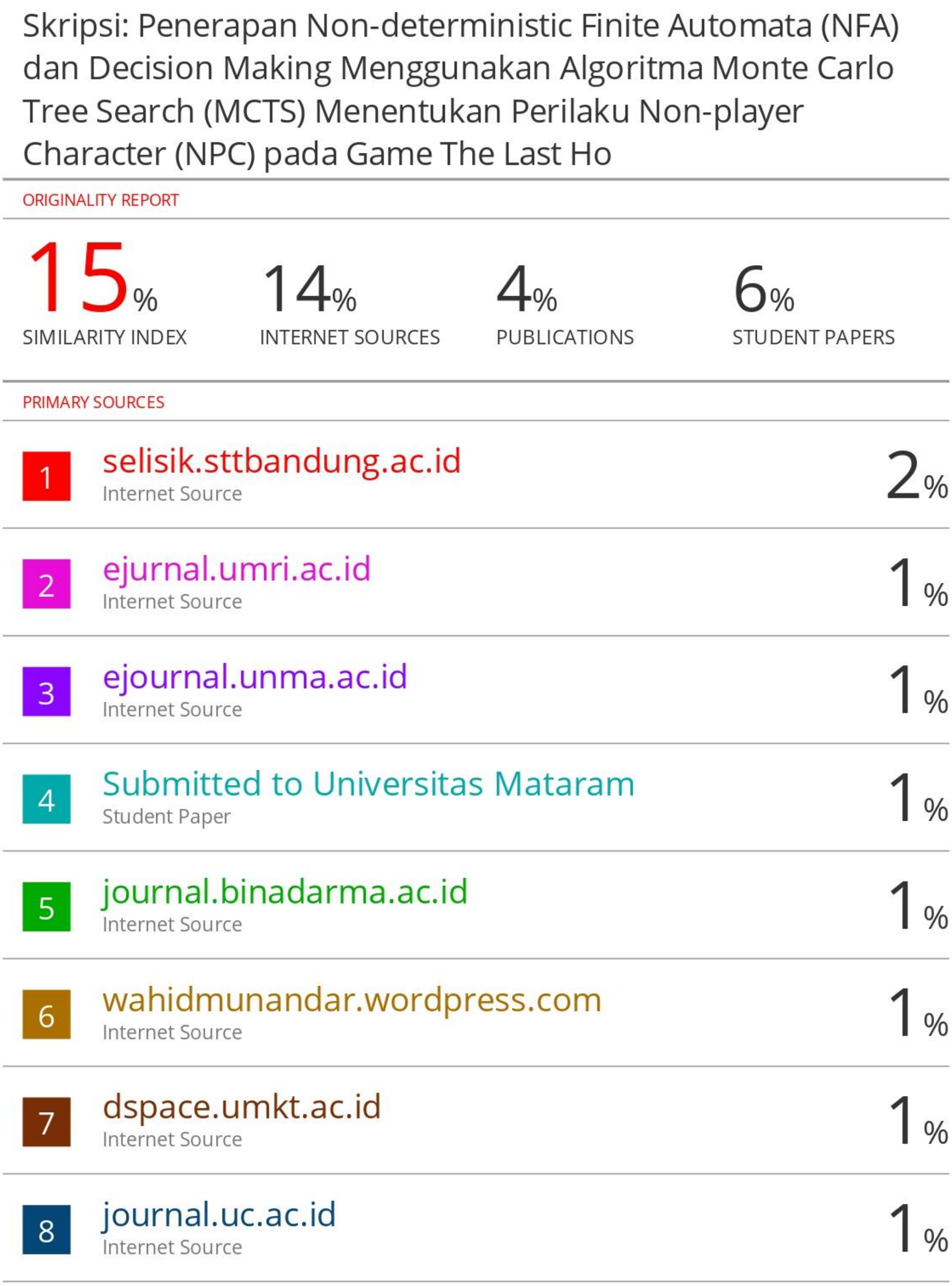
Submission date: 21-Jul-2023 11:05AM (UTC+0800)

Submission ID: 2134351649

File name: REVIE_DANIAL_PRAMADYA_-SEMINAR_HASIL.docx (6.27M)

Word count: 5825

Character count: 35496



Skripsi: Penerapan Non-deterministic Finite Automata (NFA) dan Decision Making Menggunakan Algoritma Monte Carlo Tree Search (MCTS) Menentukan Perilaku Non-player Character (NPC) pada Game The Last Ho

ORIGINALITY REPORT

15%	14%	4%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	selisik.sttbandung.ac.id Internet Source	2%
2	ejurnal.umri.ac.id Internet Source	1%
3	ejournal.unma.ac.id Internet Source	1%
4	Submitted to Universitas Mataram Student Paper	1%
5	journal.binadarma.ac.id Internet Source	1%
6	wahidmunandar.wordpress.com Internet Source	1%
7	dspace.umkt.ac.id Internet Source	1%
8	journal.uc.ac.id Internet Source	1%