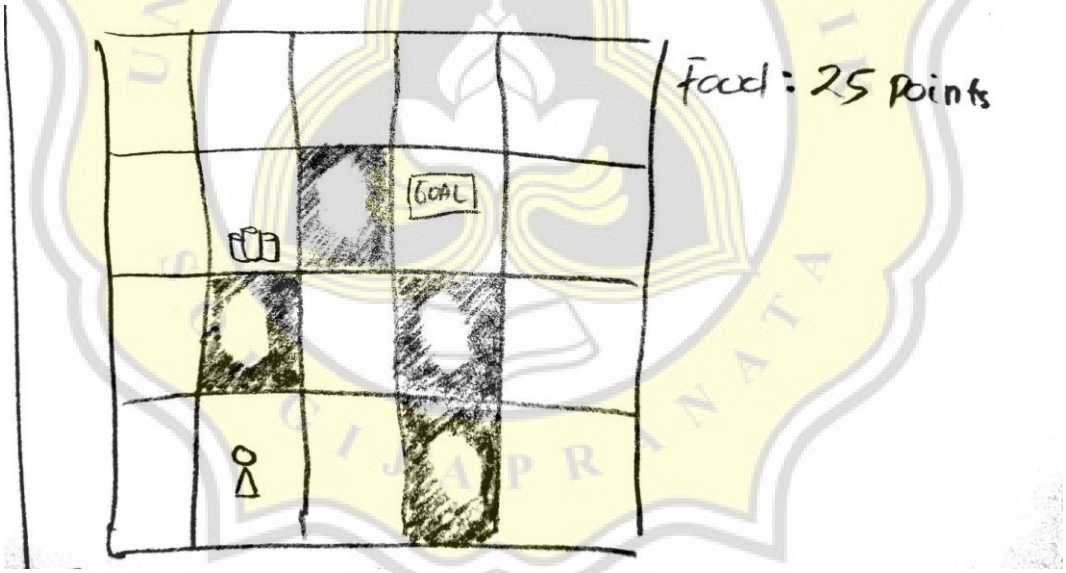
BAB IV

HASIL PENELITIAN DAN PEMBAHASAN

IV. 1 Implementasi CT

Pada game “Take Me Home” memiliki tujuan untuk memperkenalkan dan mengasah kemampuan berpikir menggunakan teknik *Computational Thinking*, untuk memenuhi tujuan tersebut perlu diketahui bahwa dalam proses perancangan pengembang harus memasukan konsep *Computational Thinking* pada desain permainan. Dalam pembuatan konsep permainan pengembang akan mengambil komponen dasar dari *CT*, Komponen ini meliputi, *Decomposition, Pattern Recognition, Abstraction, Algorithm Design,* dan *Evaluation.*

4.1.1 CT Scenario



Gambar 4.1 Scenario implementasi CT

Untuk menjelaskan implementasi CT pada game “Take Me Home” Gambar 4.1 merupakan referensi skenario masalah yang akan diselesaikan menggunakan CT.

1. *Decomposition*

   Pada Gambar 4.1 Pemain akan melakukan observasi untuk mendapatkan informasi mengenai apa saja yang ada pada area permainan. Informasi ini berupa, Posisi karakter pemain, Posisi Objektif, Posisi *Obstacle*, Jumlah *Obstacle*, Posisi *Food Item*, Jumlah *Food Item*, dan Sisa *Food Point*.

2. *Pattern Recognition*

   Dari Gambar 4.1 pemain harus mengetahui dan dapat memisahkan setiap jenis *sprite obstacle*, *spite food*, *sprite* karakter, dan *sprite objektif*.

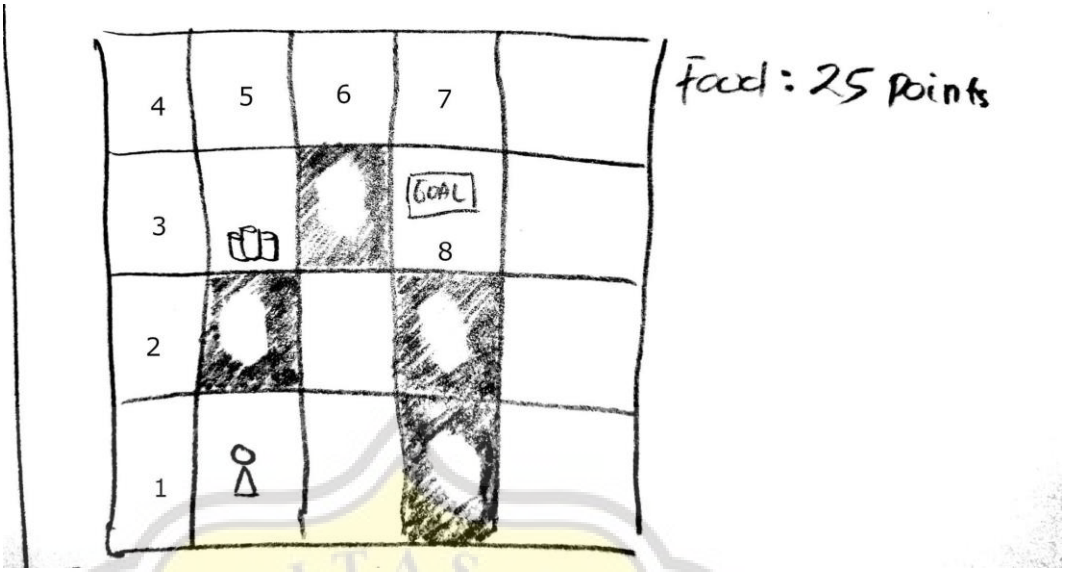
3. *Abstraction*

   Pada tahap abstraction pemain harus melihat permasalahan dari skenario Gambar 4.1 dan mengolahnya menjadi lebih sederhana. Permasalahan yang ada pada skenario Gambar 4.1 dapat diolah menjadi,

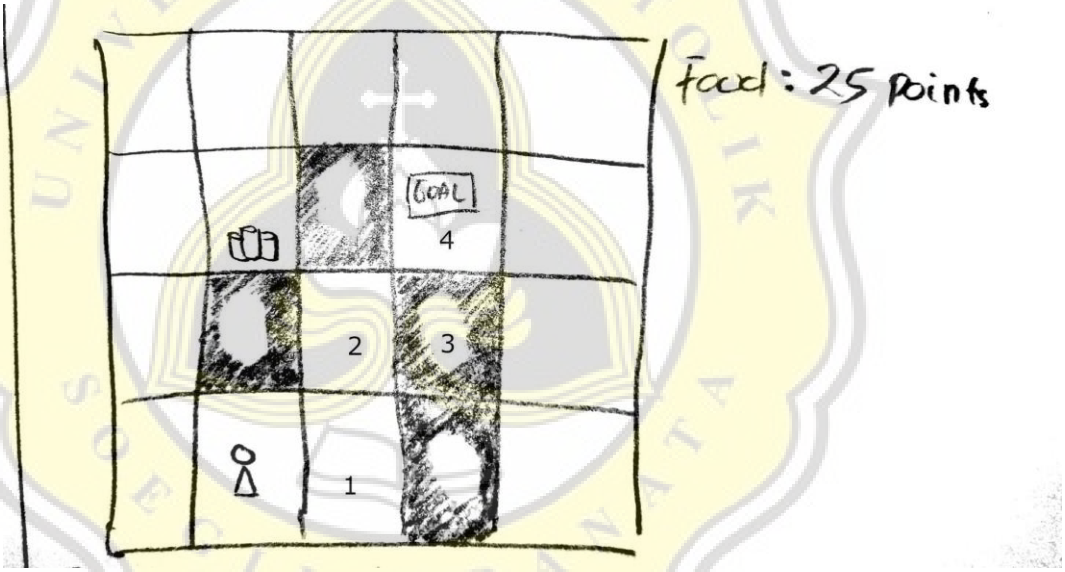
   - Memandu karakter pemain ke objektif.
   - Sisa dari poin makanan hanya 25 poin.
   - 1 step memiliki harga 5 poin makanan.
   - Menghancurkan penghalang juga memiliki harga 5 poin makanan.
   - Item makanan memberikan 25 poin makanan.
   - Mencari jalan yang memiliki harga terendah.
   - Memiliki surplus makanan untuk melanjutkan level selanjutnya.

4. *Algorithm Design*

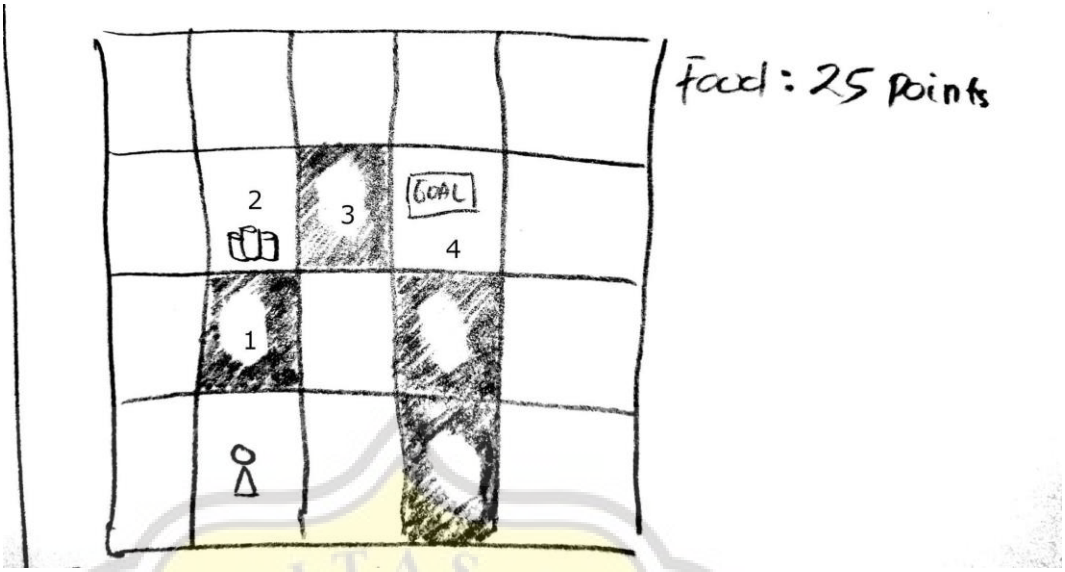
   Setelah pemain melakukan observasi dan mengetahui permasalahan yang dimiliki maka pemain harus menentukan bagaimana cara menyelesaikan skenario Gambar 4.1, berikut beberapa contoh jalur yang sekiranya akan dilihat oleh pemain,



Gambar 4.2 Jalur 1



Gambar 4.3 Jalur 2



Gambar 4.4 Jalur 3

Gambar 4.2 hingga 4.4 merupakan contoh jalur yang sekiranya dilihat oleh pemain, dari contoh diatas pemain harus mencari jalur paling sedikit memakan poin makanan dan akan memiliki sisa makanan ketika menyelesaikan skenario.

Pada contoh Gambar 4.2 pemain dapat melihat jalur tanpa harus menghancurkan penghalang, akan tetapi jalan ini memerlukan 8 langkah untuk sampai ke objektif, dari permasalahan sebelumnya 1 langkah memiliki harga 5 poin makanan maka untuk 8 langkah pemain memerlukan 40 poin makanan untuk sampai ke objektif, dengan demikian contoh Gambar 4.2 tidaklah memungkinkan untuk pilih dikarenakan sisa dari poin makanan hanya tersisa 25 poin.

Pada contoh Gambar 4.3 pemain memilih untuk menghancurkan 1 penghalang, untuk menghancurkan 1 penghalang pemain membutuhkan 5 poin makanan dan pemain memerlukan 4 langkah untuk sampai ke tempat objektif, dengan permasalahan tersebut pemain membutuhkan 25 poin makanan untuk sampai ke objektif, pemilihan jalur ini dapat dipilih oleh pemain akan tetapi ketika pemain sampai ke objektif dan melanjutkan stage selanjutnya pemain tidak memiliki sisa poin makanan dengan demikian contoh Gambar 4.3 tidak dapat menyelesaikan masalah.

Contoh Gambar 4.4 merupakan jalur optimal, pada jalur ini pemain akan mengambil *item* makanan dan menghancurkan 2 buah penghalang, tiap penghalang memerlukan 5 poin makanan dan tiap langkah memerlukan 5 makanan dengan demikian harga dari 2 buah penghalang dan 4 langkah adalah 30 poin makanan, dengan mengambil *item* makanan pemain akan mendapatkan 25 poin makanan dengan demikian pemain memiliki sisa 20 poin makanan ketika pemain sampai ke objektif.

Dari 3 contoh jalur yang dapat dilihat pemain contoh Gambar 4.4 merupakan jalur terbaik, maka pemain juga perlu menuliskan jalur yang harus dibuat menggunakan *command* agar karakter pemain dapat bergerak menuju ke objektif.

5. *Evaluation*

Ketika pemain telah selesai menuliskan *Command* pemain akan melihat karakter pemain bergerak sesuai dengan *Command* yang telah diberikan, jika pemain melakukan kesalahan ketika menulis *Command* pemain akan melihat karakter pemain tidak berada pada tempat yang telah pemain prediksi maka dari itu pemain harus melakukan observasi ulang dan melihat jalur alternatif untuk menyelesaikan skenario dengan sisa point makanan yang tersisa.

IV. 2 Perancangan Game

"Take Me Home" adalah *game* yang dirancang sebagai prototipe atau *proof of concept* untuk mengenalkan konsep dasar dari pemahaman berpikir secara komputasional. *Game* ini memiliki perspektif 2D dan merupakan tipe *endless game* yang dimana pemain dapat memainkan *game* ini secara terus menerus. Dalam *game* ini pemain diharuskan untuk menyelesaikan puzzle untuk melanjutkan stage berikutnya, setiap stage memiliki pola yang sama yakni pemain perlu membuat kumpulan perintah untuk menuntun karakter pemain agar dapat keluar dari stage tersebut. Agar game ini tetap menarik dan tidak membosankan "Take Me Home" menggunakan *procedural generated stage* yang dimana setiap stage akan selalu

berbeda, semakin banyak stage yang dilewati juga akan menambah tingkat kesulitan stage.

IV. 3 Game Design

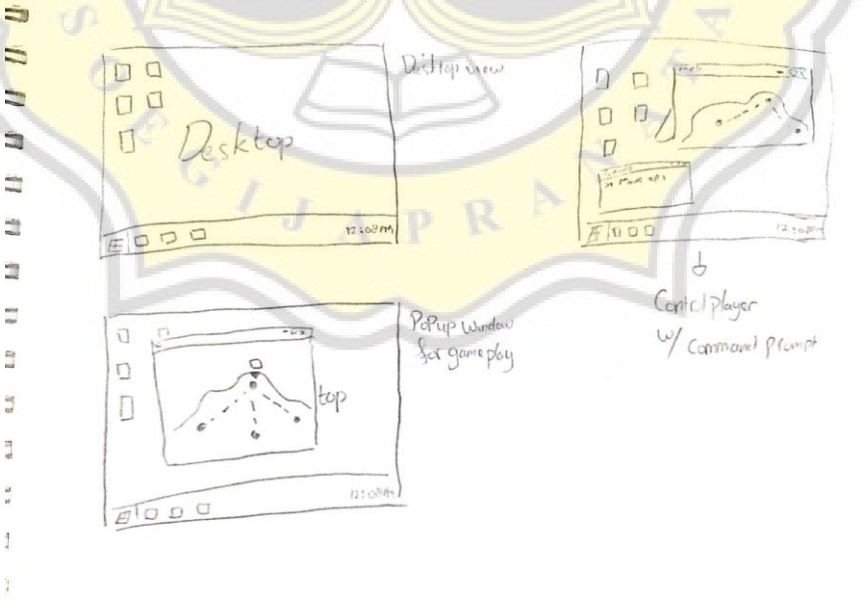
"Take Me Home" merupakan *game* 2D yang memiliki perspektif *Top Down View*, dan memiliki tipe *Endless Game*, *Typing Game*, dan *Procedural Generated*. Dengan menggunakan *command prompt* pemain perlu menuntun karakternya agar dapat keluar dari stage tersebut.

IV. 4 Pre production

Masa *Pre - Production* merupakan masa dimana pengembang permainan mengumpulkan data, ide, dan membuat mockup dari sebuah produk. Pada masa ini juga pengembang sudah menetapkan bagaimana bentuk akhir dari produk yang akan dibuat.

IV.6.1 Desain awal

"Take Me Home" memiliki beberapa desain yang berbeda ketika masa perancangan dimulai. Perbedaan ini berada pada desain keseluruhan baik secara *gameplay* maupun desain visual.



Gambar 4.5 "Take Me Home" Tampilan desain awal

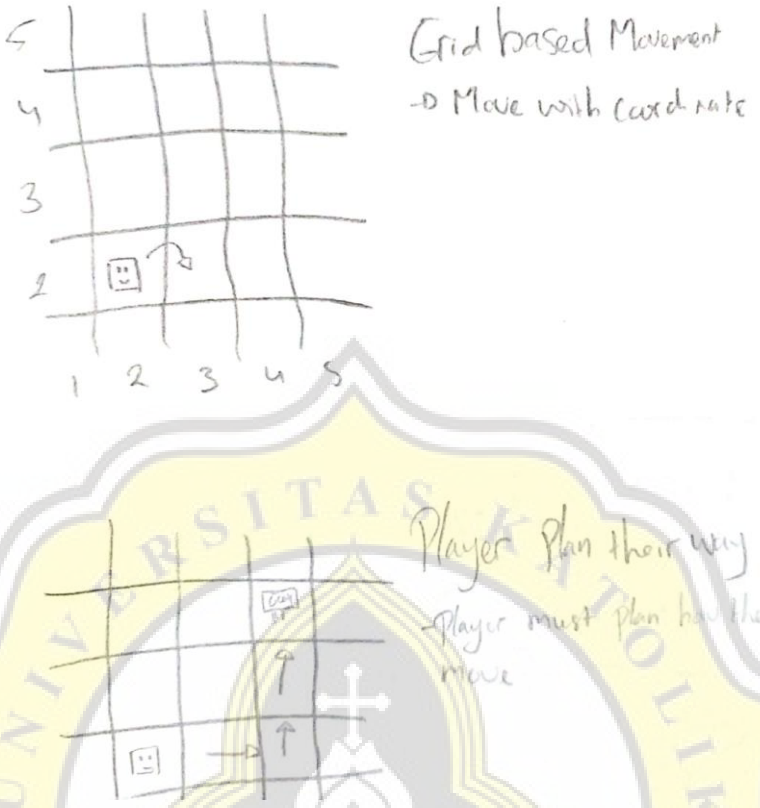
"Take Me Home" awalnya memiliki desain visual layaknya sebuah komputer, layar game akan berbentuk layaknya desktop dan juga memiliki tombol aplikasi yang dapat dibuka. Setiap aplikasi merupakan elemen dari *gameplay*.

Gambar 4.6 "Take Me Home" Fitur desain awal

Pada desain awal "Take Me Home" memiliki tipe game *roguelike* yang dimana pemain hanya memiliki 1 nyawa dan jika kalah progress pemain dihapus. Fitur yang ada pada desain awal mencakup fitur game *roguelike* pada umumnya seperti *procedural generated map, random loot/ item, enemy encounter,* bertarung, dan hanya memiliki 1 nyawa. Desain ini memiliki fitur *command prompt* yang dimana pemain akan terus menggunakan fitur ini untuk mengontrol tindakan karakter di dalam game.

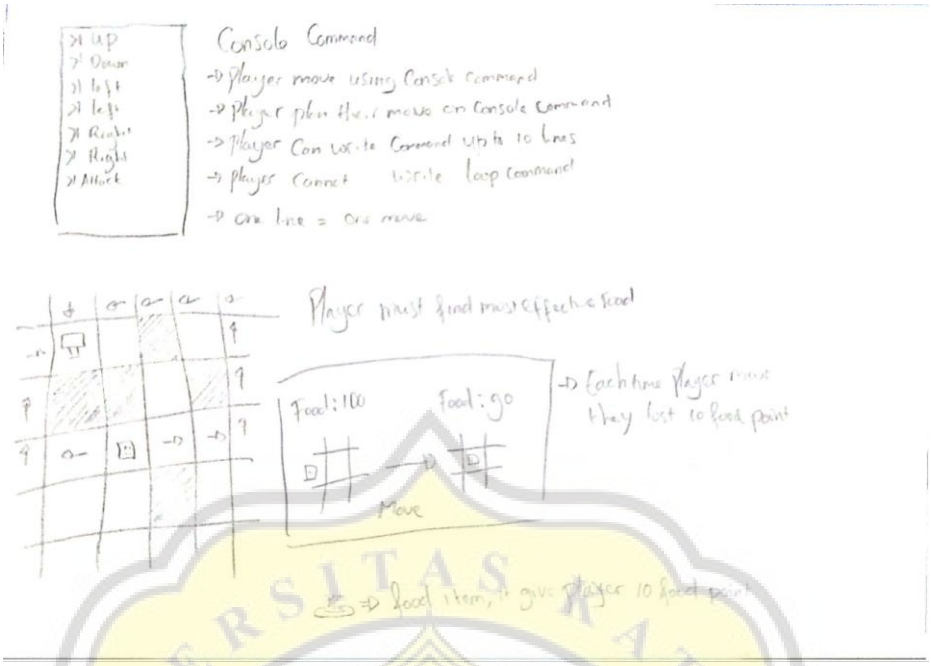
IV.6.2 Desain akhir

Desain akhir dari "Take Me Home" memiliki *scope* yang lebih sederhana dibandingkan desain awal, hal ini dikarenakan adanya batasan kemampuan dan kurangnya aspek yang ingin diberikan.



Gambar 4.7 "Take Me Home", grid movement

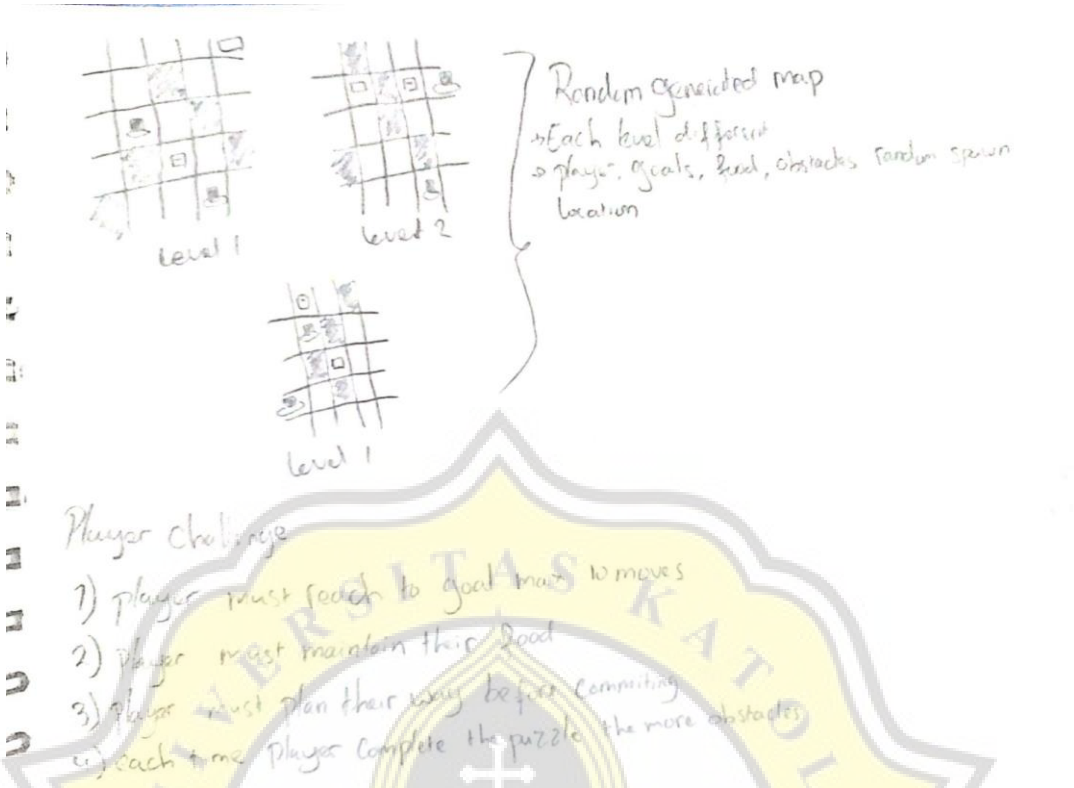
Pada desain akhir area permainan dibuat lebih simpel dengan menggunakan *grid type area*. Tujuan pemain juga dibuat lebih sederhana yakni menyelesaikan puzzle dengan menuntun karakter sampai ke tujuan dengan menggunakan *console command*.



Gambar 4.8 "Take Me Home", Console Command dan Food Point

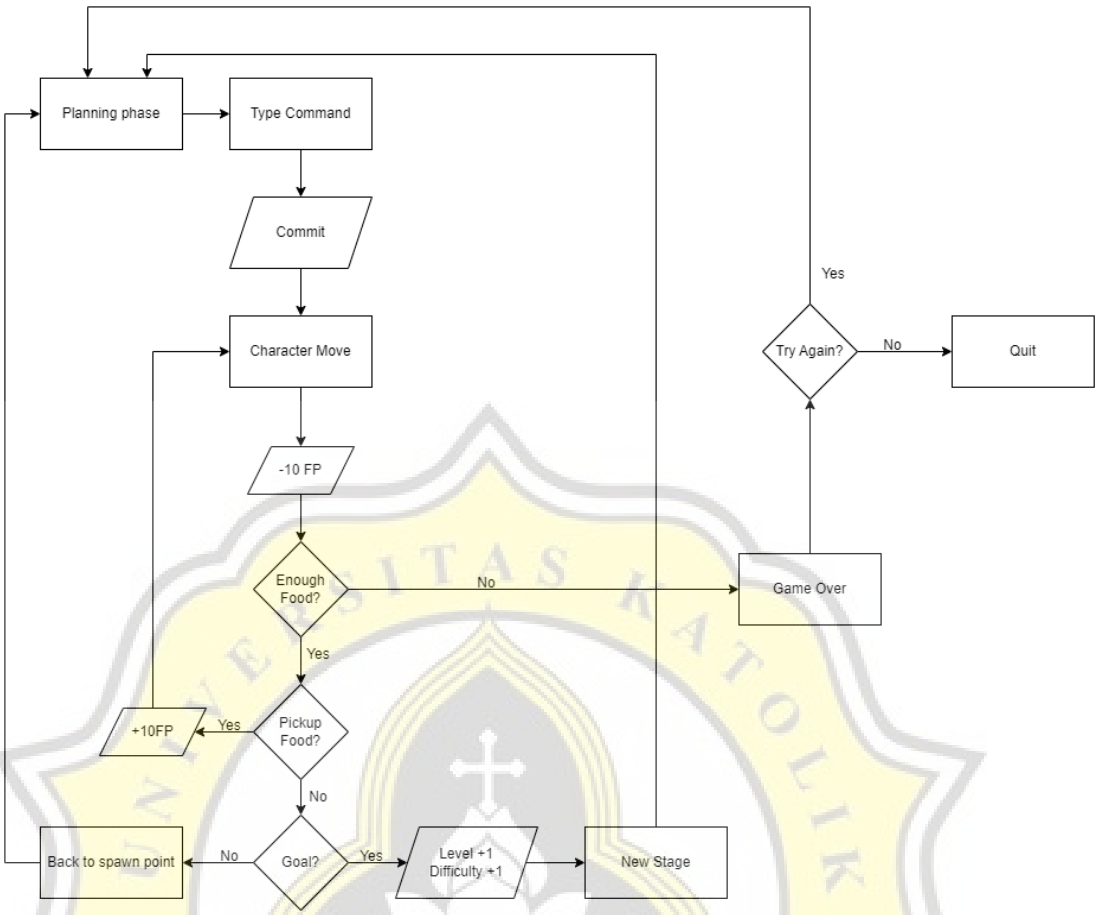
Penggunaan *console command* tidak dihapus akan tetapi diubah, pada desain awal input *command* dibuat layaknya *command prompt* pada komputer, *console command* pada desain akhir dibuat lebih simpel dan menggunakan *command* yang lebih sederhana. *Console command* pada desain terakhir hanya memiliki 10 baris kode, pemain dapat mengetik *command* secara bertumpuk dan perlu melakukan *commit* untuk menjalankan *command*.

Untuk menambah tantangan, terdapat variabel bernama *Food Point*. Variabel ini berfungsi sebagai *Resource* pemain untuk bergerak, tiap kali pemain bergerak variabel ini akan berkurang dan ketika variabel ini habis maka game akan selesai. Pada area permainan pemain terdapat item bernama *Food* yang memiliki fungsi untuk menambah *Food Point* milik pemain. Dengan menyeimbangan variabel ini pemain dapat mengatur dan memilih jalur yang dianggap paling efektif dan dapat memainkan permainan terus menerus.



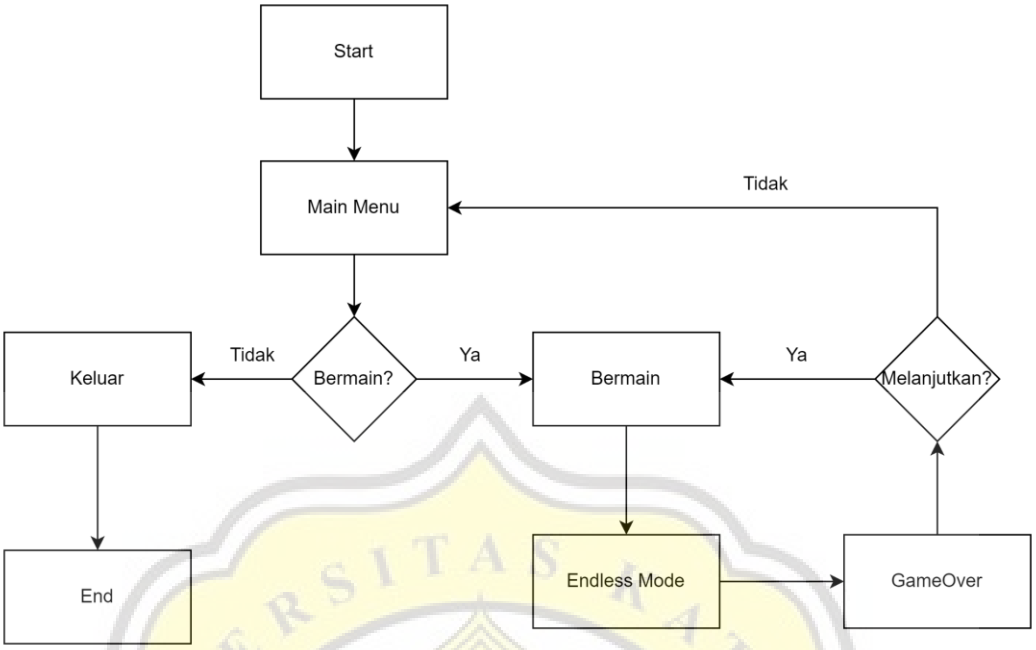
Gambar 4.9 "Take Me Home", Procedural generated dan challenge

Untuk membuat pemain tidak cepat merasa bosan area permainan dibuat secara acak dan *procedural*, dengan menggunakan cara ini setiap level permainan dapat menjadi tak terbatas dan berbeda tiap levelnya. Proses pengacakan ini mengacak lokasi tujuan pemain, tempat respawn pemain, tempat *Food* muncul, dan lokasi dinding penghalang.



Gambar 4.10 Core Loop "Take Me Home"

Pada gambar 4.10 *game loop* dari "Take Me Home" cukup sederhana, dimulai dari pemain memikirkan rute terdekat dan paling efisien yang kemudian diterjemahkan menggunakan *command* pemain dapat memandu karakter dalam *game* agar dapat sampai ke *goal.* "Take Me Home" memiliki fokus pada mengembangkan dan memperkenalkan konsep dasar dari *Computational Thinking* yang dimana pada *core loop* "Take Me Home" pemain akan belajar bagaimana cara menemukan permasalahan, mengamati keadaan, memanfaatkan peralatan yang ada pada masa *planning*, dan juga menyelesaikan masalah dengan merancang sebuah algoritma pada *Type Command.*



Gambar 4.11 Game Flow

Perancangan *gameplay* "Take Me Home" lebih difokuskan pada *endless challenge* yang dimana pemain dapat bermain bebas. Tampilan menu awal dibuat sangat simpel dengan memberikan tombol bermain dan keluar, tutorial permainan langsung diterapkan pada layar permainan ketika pemain ingin bermain.

IV. 5 Implementasi Pembuatan Game

Untuk implementasi pembuatan *game* akan dijelaskan tentang kondisi menyelesaikan *game*, tutorial dan *gameplay* sesuai dengan perancangan sebelumnya.

IV.6.1 Implementasi Tutorial

Pada awal memulai permainan pemain akan dilihatkan cara bermain dari *game* "Take Me Home", tampilan dari petunjuk ini berupa *popup slide show* tutorial yang akan muncul ketika pemain menekan tombol *Help* atau mengetik *Help* pada command prompt.

Gambar 4.12 Popup Tutorial

Pada gambar 4.12 ditampilkan penjelasan aspek-aspek apa saja yang terdapat pada *game* dan bagaimana cara bermain. Pemain dapat menutup *popup* bantuan dengan cara mengetikan perintah *Help* pada command prompt atau menekan tombol *close* pada bagian kanan atas panel *popup*.

Panel *popup* bantuan menggunakan model *slide show*, pada Gambar 4.13 ditampilkan tombol untuk menggeser gambar tutorial, untuk mengganti ke halaman selanjutnya pemain dapat menekan tombol yang ada di bagian kanan gambar dan jika pemain ingin melihat gambar sebelumnya pemain dapat menekan tombol yang ada di bagian kiri gambar. Tombol sengaja dibuat tidak terlihat agar tampilan *popup* terlihat lebih bersih dan gambar dapat terlihat dengan jelas, ketika pemain mengarahkan *mouse pointer* ke lokasi tombol, tombol akan muncul berupa *highlight*.

Gambar 4.13 Tutorial Next dan Previous button

```
public class HelpButton : MonoBehaviour
{

    public GameObject[] helpPanels;

    public Console manager_Console;

    int currentPanel = 0;


    public void NextSlide(){
        if(currentPanel < helpPanels.Length - 1){
            currentPanel++;
            helpPanels[currentPanel - 1].SetActive(false);
            helpPanels[currentPanel].SetActive(true);
        }
    }


    public void PreviousSlide(){
        if(currentPanel > 0){
            currentPanel--;
            helpPanels[currentPanel + 1].SetActive(false);
            helpPanels[currentPanel].SetActive(true);
        }
    }


    public void CloseHelpMenu(){
        manager_Console.HelpControl();
    }
}
```

Gambar 4.14 Script HelpButton

Gambar 4.14 adalah skrip yang dipakai untuk mengontrol halaman pada *popup* tutorial. Fungsi yang digunakanan pada skrip ini adalah fungsi mengaktifkan atau menonaktifkan *game* objek.

```
public void HelpControl(){

    helpMenu.gameObject.SetActive (!helpMenu.gameObject.activeSelf);
    helpMenuOpen = helpMenu.gameObject.activeSelf;
    helpSplash.gameObject.SetActive(helpMenuOpen);

    sideHelpUI.gameObject.SetActive(!helpMenuOpen);

}
```

Gambar 4.15 fungsi untuk mengontrol popup help

Gambar 4.15 merupakan bagian dari fungsi yang terdapat pada skrip *console.cs*. Fungsi ini digunakan untuk mengaktifkan atau menonaktifkan *game object help menu*.

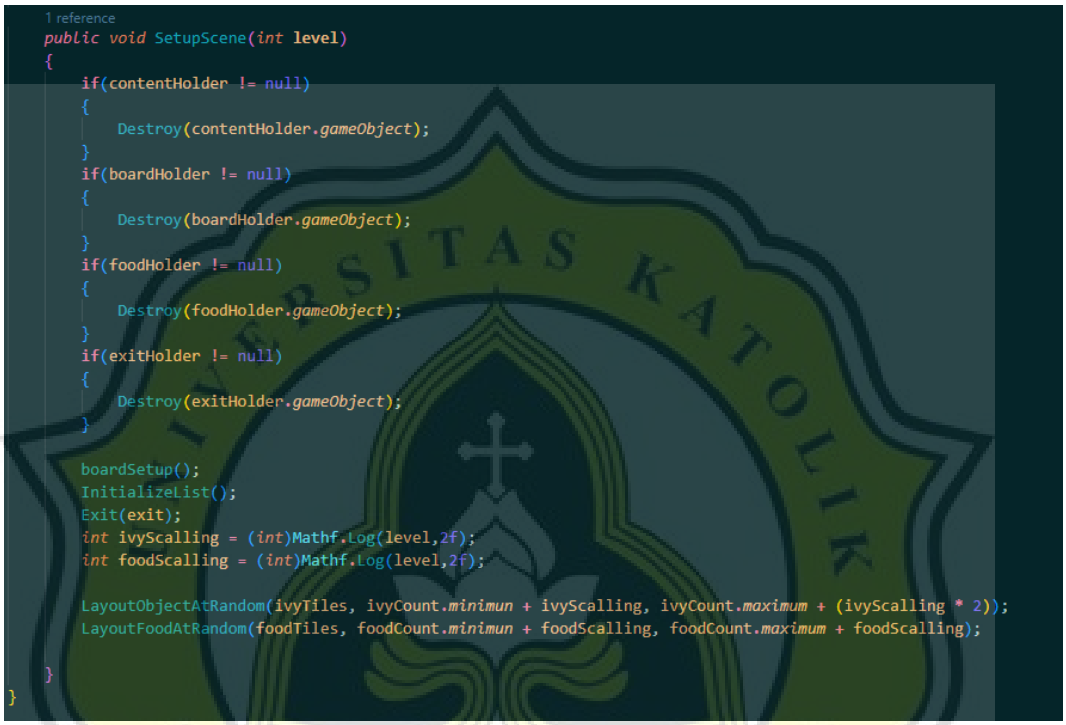
Gambar 4.16 Tampilan setelah popup bantuan ditutup

Ketika *popup* bantuan ditutup pemain akan melihat area permainan (tengah), konsole (kiri), tombol bantuan dan list perintah (kanan). Pada tampilan ini pemain sudah dapat memainkan permainan.

IV.6.2 Implementasi Gameplay

- *Map Generator*

Pada game "Take Me Home" area permainan dibuat secara otomatis menggunakan *map generator*, penggunaan fungsi ini ditujukan untuk membuat area permainan berbeda-beda dengan kombinasi tanpa batas.



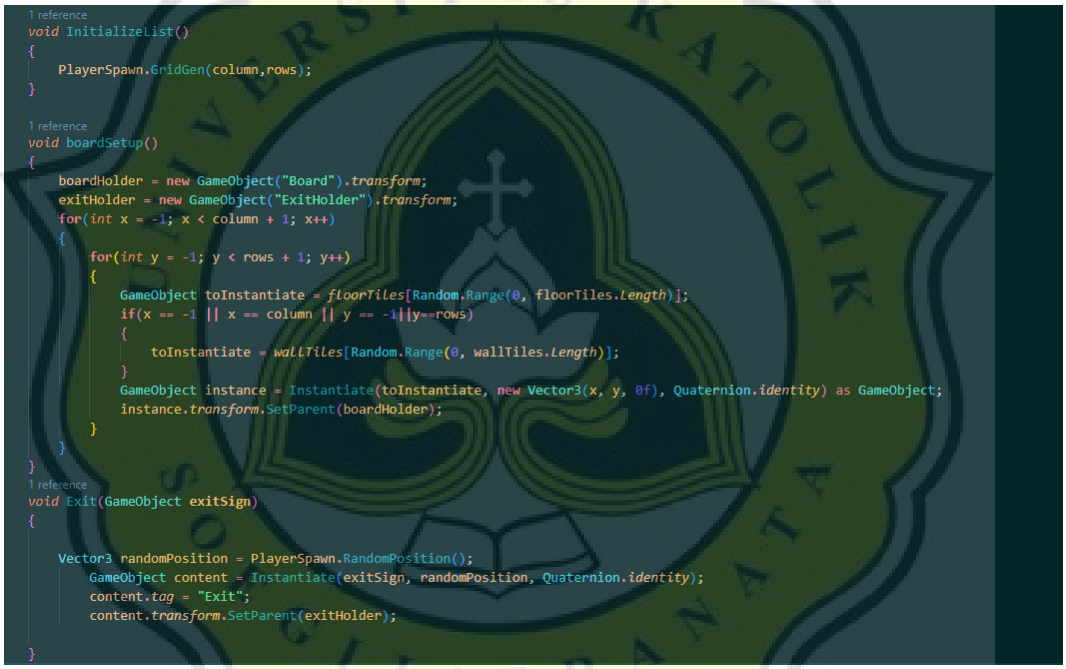
```
1 reference
public void SetupScene(int level)
{
    if(contentHolder != null)
    {
        Destroy(contentHolder.gameObject);
    }
    if(boardHolder != null)
    {
        Destroy(boardHolder.gameObject);
    }
    if(foodHolder != null)
    {
        Destroy(foodHolder.gameObject);
    }
    if(exitHolder != null)
    {
        Destroy(exitHolder.gameObject);
    }

    boardSetup();
    InitializeList();
    Exit(exit);
    int ivyScalling = (int)Mathf.Log(level,2f);
    int foodScalling = (int)Mathf.Log(level,2f);

    LayoutObjectAtRandom(ivyTiles, ivyCount.minimun + ivyScalling, ivyCount.maximum + (ivyScalling * 2));
    LayoutFoodAtRandom(foodTiles, foodCount.minimun + foodScalling, foodCount.maximum + foodScalling);

}
}
```

Gambar 4.17 Fungsi untuk membuat area permainan

Gambar 4.17 merupakan fungsi yang terdapat pada *BoardManager.cs* yang digunakan untuk membentuk area permainan. Fungsi ini memanggil fungsi - fungsi lainnya seperti fungsi *boardSetup(), InitializeList(), Exit(), LayoutObjectAtRandom()*, dan fungsi *LayoutFoodAtRandom()*. Fungsi - fungsi ini digunakan untuk membuat grid dan meletakan *prefabs* yang ada.

```
5 references
public static List<Vector3> gridPosition = new List<Vector3>();

1 reference
public static void GridGen(int column, int rows){
    gridPosition.Clear();

    for(int x = 1; x < column; x++)
    {
        for(int y = 1; y < rows; y++)
        {
            gridPosition.Add(new Vector3(x, y, 0f));
        }
    }
}
```

Gambar 4.18 Fungsi untuk membentuk dan menyimpan koordinat grid



```
1 reference
void InitializeList()
{
    PlayerSpawn.GridGen(column,rows);
}

1 reference
void boardSetup()
{
    boardHolder = new GameObject("Board").transform;
    exitHolder = new GameObject("ExitHolder").transform;
    for(int x = -1; x < column + 1; x++)
    {
        for(int y = -1; y < rows + 1; y++)
        {
            GameObject toInstantiate = floorTiles[Random.Range(0, floorTiles.Length)];
            if(x == -1 || x == column || y == -1||y==rows)
            {
                toInstantiate = wallTiles[Random.Range(0, wallTiles.Length)];
            }
            GameObject instance = Instantiate(toInstantiate, new Vector3(x, y, 0f), Quaternion.identity) as GameObject;
            instance.transform.SetParent(boardHolder);
        }
    }
}
1 reference
void Exit(GameObject exitSign)
{

    Vector3 randomPosition = PlayerSpawn.RandomPosition();
        GameObject content = Instantiate(exitSign, randomPosition, Quaternion.identity);
        content.tag = "Exit";
        content.transform.SetParent(exitHolder);

}
```

Gambar 4.19 Fungsi InitializeList(), boardSetup() dan Exit()

Pada Gambar 4.19 terdapat fungsi *InitializeList()* yang digunakan untuk memanggil fungsi yang terdapat pada skrip *PlayerSpawn.cs*, Fungsi yang dipanggil merupakan fungsi yang terdapat pada Gambar 4.18 pada fungsi ini skrip akan membuat sebuah *array* yang berisikan *grid position* berupa *Vector3* yang besar X dan Y disesuaikan dengan *column* dan *rows* yang sudah ditentukan pada *inspector*.

Terdapat pula fungsi *boardSetup()* yang digunakan untuk mengacak jenis *prefabs floor* dan meletakkannya pada lokasi berdasarkan *grid list* yang telah dibuat

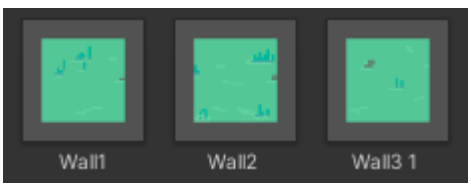
pada fungsi *InitializeList()*, fungsi ini juga membuat *game object* baru bernama *boardHolder* dan *exitHolder*, *game object* ini berfungsi untuk menyimpan *prefabs* yang sudah di instantiate agar pada *hierarchy* lebih rapi, peletakan pembatas juga terjadi pada fungsi ini dengan meletakan *prefabs Walls* pada area permainan.

Exit() memiliki fungsi yang sama dengan fungsi *boardSetup()* akan tetapi pada fungsi ini skrip akan meletakan *prefabs Exit* dengan lokasi yang diacak berdasarkan *list grid* yang sudah ada.

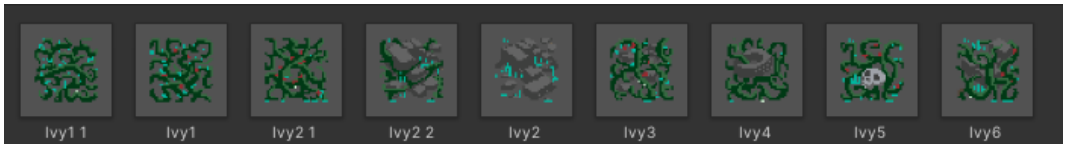
```
1 reference
void LayoutObjectAtRandom(GameObject[] tilesArray, int minimun, int maximum)
{
    int objectCount = Random.Range(minimun, maximum + 1);
    contentHolder = new GameObject("ContentHolder").transform;
    for(int i = 0; i < objectCount; i++)
    {
        Vector3 randomPosition = PlayerSpawn.RandomPosition();
        GameObject tileChoice = tilesArray[Random.Range(0,tilesArray.Length)];
        GameObject content = Instantiate(tileChoice, randomPosition, Quaternion.identity);
        content.tag = "Content";
        content.transform.SetParent(contentHolder);
    }
}

1 reference
void LayoutFoodAtRandom(GameObject[] tilesArray, int minimun, int maximum)
{
    int objectCount = Random.Range(minimun, maximum + 1);
    foodHolder = new GameObject("FoodHolder").transform;
    for (int i = 0; i < objectCount; i++)
    {
        Vector3 randomPosition = PlayerSpawn.RandomPosition();
        GameObject tileChoice = tilesArray[Random.Range(0, tilesArray.Length)];
        GameObject foodObject = Instantiate(tileChoice, randomPosition, Quaternion.identity);
        foodObject.transform.SetParent(foodHolder);
    }
}
```

Gambar 4.20 Fungsi LayoutObjectAtRandom() dan LayOutFoodAtRandom()

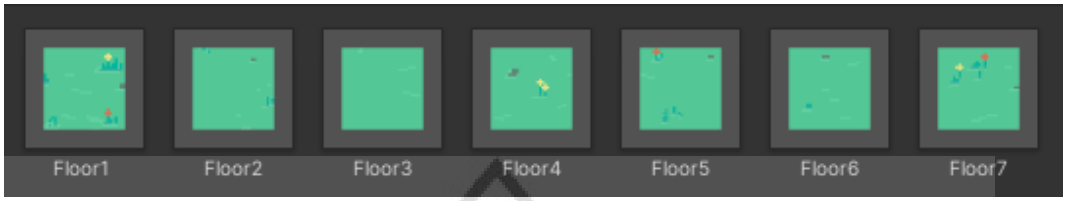
Fungsi pada Gambar 4.20 mengatur lokasi peletakan objek penghalang dan objek makanan. Pada fungsi ini lokasi peletakan akan diacak berdasarkan *list* yang ada pada *grid list* awal dan lokasi yang telah dipakai akan dihapuskan agar tidak terjadi penumpukan *object* pada lokasi yang sama, *prefabs* yang telah di instantiate akan dimasukan ke *object parent* bernama *contentHolder* dan *foodHolder*.



Gambar 4.21 Prefabs Walls



Gambar 4.22 Prefabs Ivy/Obstacles



Gambar 4.23 Prefabs Floors

Gambar 4.24 Prefabs Foods

Gambar 4.21 hingga 4.24 merupakan *prefabs* yang digunakan pada *game* "Take Me Home", *prefabs* ini akan dipilih dan diletakan secara acak pada fungsi-fungsi yang ada pada Gambar 4.19 dan Gambar 4.20.

- *Game Manager*

Game manager merupakan sebuah *game object* yang digunakan untuk menampung skrip - skrip penting agar permainan dapat berjalan tanpa ada kendala. Fungsi dari *game manager* ini juga sebagai satu object yang dapat mengatur dan dapat dipanggil oleh skrip lain.

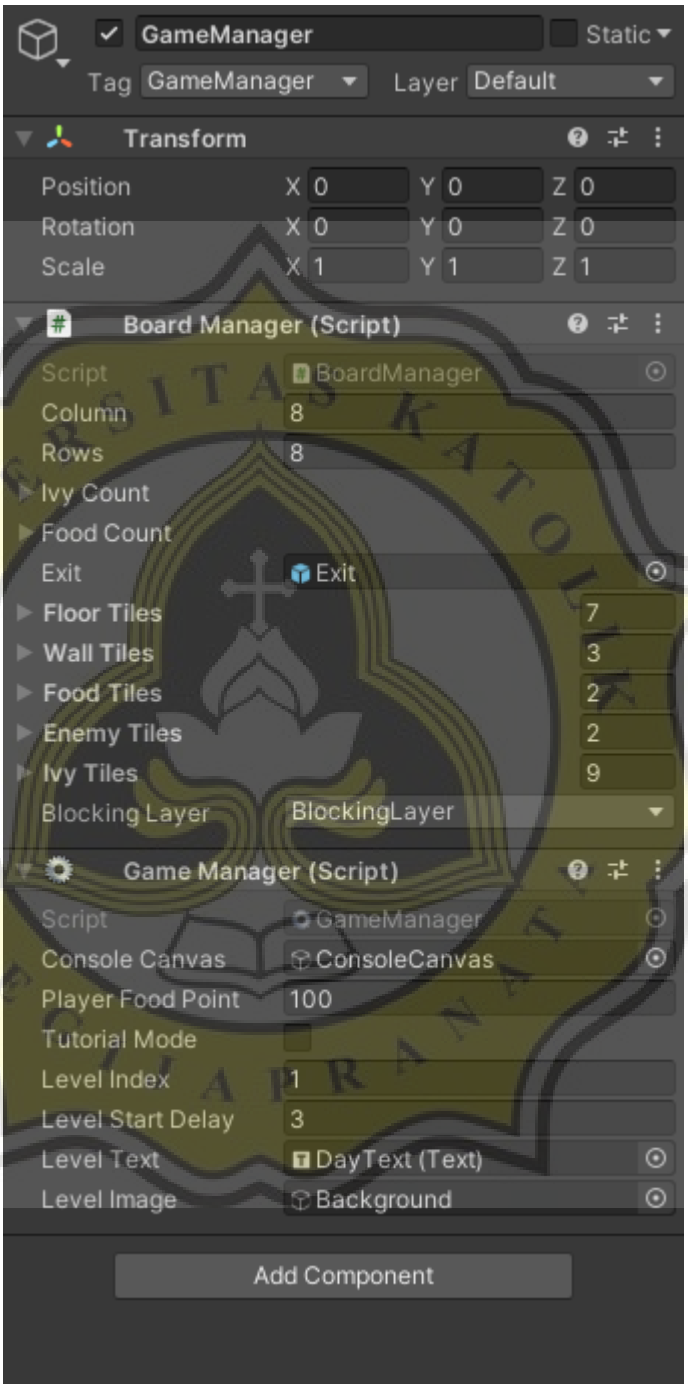
```
using UnityEngine;
using UnityEngine.UI;

6 references
public class GameManager : MonoBehaviour
{
    1 reference
    TutorialManagerNew Tmanager;

    4 references
    public GameObject consoleCanvas;
    4 references
    public static GameManager instance = null;
    3 references
    BoardManager boardScript;
    2 references
    public int playerFoodPoint = 100;
    3 references
    public bool tutorialMode = false;

    3 references
    public int levelIndex = 1;
    1 reference
    public float levelStartDelay = 2f;

    //[SerializeField] private GameObject player;
    //[SerializeField] private GameObject movePoint;
    2 references
    public Text levelText;
    3 references
    public GameObject levelImage;
    2 references
    private bool doingSetup;

    0 references | Unity Message
    void Awake()
    {
        if(!tutorialMode){

            if (instance == null)
            {
                instance = this;
            }
            else if(instance != this)
            {

            }

        }

        InitGame();
```

Gambar 4.25 GameManager.cs

```
        1 reference
44      public void FindComponent(){
45          boardScript = GetComponent<BoardManager>();
46          consoleCanvas = GameObject.FindGameObjectWithTag("Console");
47
48      }
49
        0 references
50      public void GameOver()
51      {
52          enabled = false;
53      }
54
        2 references
55      public void InitGame()
56      {
57          doingSetup = true;
58          FindComponent();
59          Invoke("HideLevelImage", levelStartDelay);
60
61          ShowLevelImage();
62          if(boardScript != null){
63              boardScript.SetupScene(levelIndex);
64          }
65
66          if(tutorialMode){
67              TutorialScene();
68          }
69
70      }
71
        0 references
72      private void HideLevelImage()
73      {
74          levelImage.SetActive(false);
75          consoleCanvas.SetActive(true);
76          doingSetup = false;
77
78          if(tutorialMode){
79              Tmanager.Tutorial();
80          }
81      }
82
        1 reference
83      private void ShowLevelImage()
84      {
85          levelImage.SetActive(true);
86          levelText.text = "Day " + levelIndex;
87          consoleCanvas.SetActive(false);
88      }
89
        1 reference
90      void TutorialScene(){
91          levelImage.SetActive(true);
92          levelText.text = "tutorial";
93          consoleCanvas.SetActive(false);
94      }
95
96  }
97
```

Gambar 4.25 GameManager.cs (lanjutan)

Gambar 4.25 merupakan skrip *GameManager*, skrip ini digunakan untuk mengatur pembuatan area, mengontrol pergantian *scene*, mengaktifkan dan menonaktifkan *game object*.



Gambar 4.26 Object Game Manager

Pada *object game manager* terdapat 2 skrip yakni *BoardManager.cs* dan *GameManager.cs*, semua *prefabs* yang ada akan diletakan pada game object ini dan akan dipanggil melalui game object ini.

- *Console* dan *Input*

Input sistem pada game "Take Me Home" tidak menggunakan sistem input yang ada, hal ini digunakan agar input yang dimasukan tidak sembarangan dan sudah tersaring.

```
1 reference
public static Command CommandFromText(string text, int lineIndex)
{
    CommandType t = CommandType.Null;
    bool attack = false;
    if(string.IsNullOrEmpty(text))
    {
        t = CommandType.Ignore;
    }
    else
    {
        if(text.Length > 0)
        {
            if(text.Contains("attack") || text[0] == 'a' || text.Contains("att"))
            {
                text = text.Replace("attack", "");
                text = text.Replace("a", "");
                text = text.Replace("att","");
                text = text.Replace(" ","");

                attack = true;
            }

            if(attack){

                switch (text[0])
                {
                    case 'u':
                        t = CommandType.UpAttack;
                        break;
                    case 'l':
                        t = CommandType.LeftAttack;
                        break;
                    case 'd':
                        t = CommandType.DownAttack;
                        break;
                    case 'r':
                        t = CommandType.RightAttack;
                        break;
                    default:
                        t = CommandType.Ignore;
                        break;
                }

            }
```

Gambar 4.27 Fungsi CommandFromText

```
            else{

                switch (text[0])
                {
                    case 'u':
                        t = CommandType.Up;
                        break;
                    case 'l':
                        t = CommandType.Left;
                        break;
                    case 'd':
                        t = CommandType.Down;
                        break;
                    case 'r':
                        t = CommandType.Right;
                        break;
                    case 'w':
                        t = CommandType.Wait;
                        break;
                    default:
                        t = CommandType.Ignore;
                        break;
                }

            }

        }
    }
    int steps = ExtractNumberFromText(text);
    return new Command(t, steps, lineIndex);
}
```

Gambar 4.28 Fungsi CommandFromText (lanjutan)

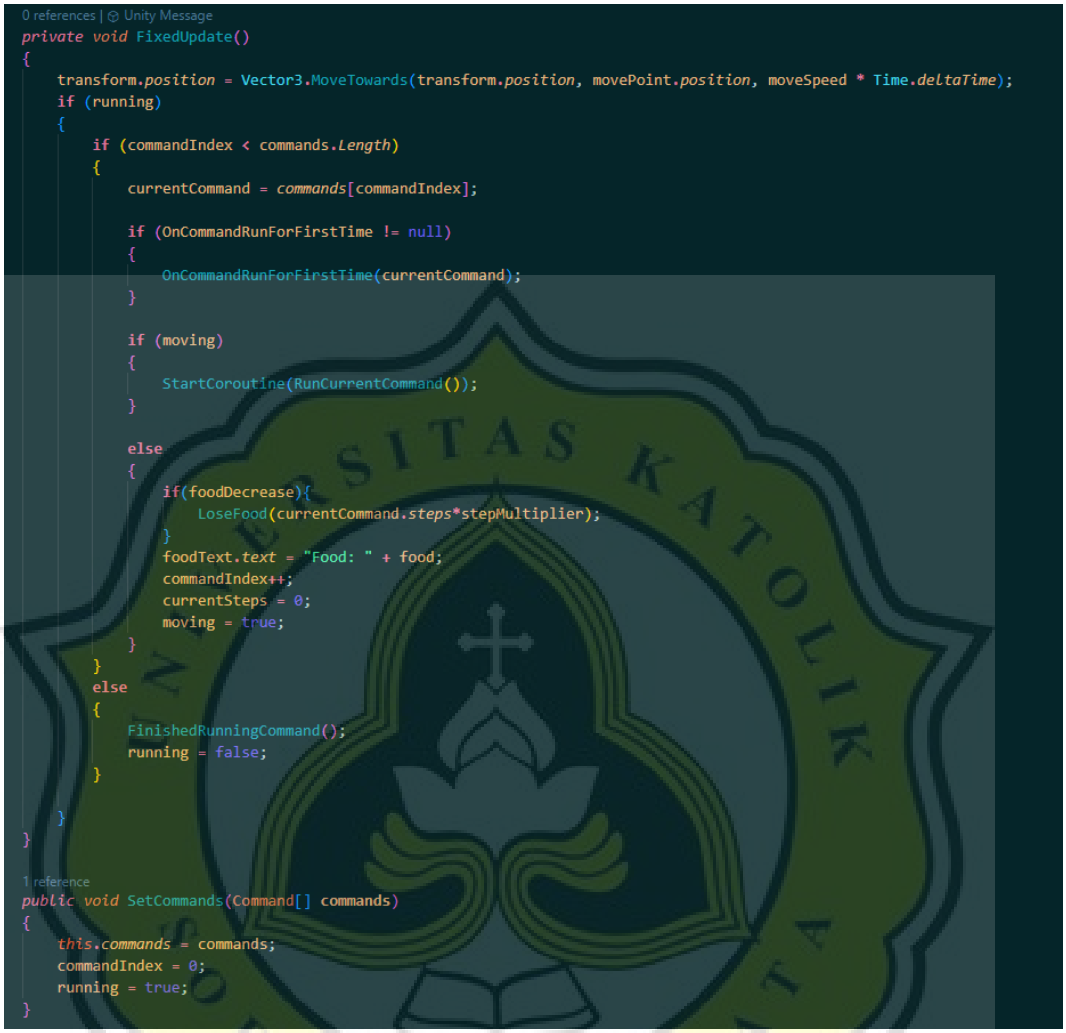
Gambar 4.28 merupakan salah satu fungsi pada skrip *Command.cs* yang dimana pada fungsi ini hasil dari penyederhanaan input di cek dan diteruskan sebagai fungsi *Command()* baru. Pada proses ini hasil penyederhanaan *string text* akan dicocokan dengan *enum CommandType* yang ada.

```
1 reference
public static Command[] CommandsFromLines(List<string> lines) //data dari console dimasukan kedalam array commands
{
    Command[] commands = new Command[lines.Count];
    for(int i = 0; i < lines.Count; i++)
    {
        commands[i] = CommandFromText(lines[i], i); //foreach data yg ada di dalam command[] dicocokan dengan switch case
    }
    return commands;
}
```

Gambar 4.29 Fungsi CommandsFromLines

Fungsi *CommandFromLines* merupakan fungsi yang digunakan untuk memisahkan dan menyederhanakan input pada *array Command*, setiap *index* yang ada pada *array Command* akan disederhanakan dan dilanjutkan pada fungsi pada

Gambar 4.28 dan hasil akhir akan dikembalikan sebagai variabel *commands* yang nantinya variabel ini akan diterima pada skrip *Player.cs*.



```
0 references | Unity Message
private void FixedUpdate()
{
    transform.position = Vector3.MoveTowards(transform.position, movePoint.position, moveSpeed * Time.deltaTime);
    if (running)
    {
        if (commandIndex < commands.Length)
        {
            currentCommand = commands[commandIndex];

            if (OnCommandRunForFirstTime != null)
            {
                OnCommandRunForFirstTime(currentCommand);
            }

            if (moving)
            {
                StartCoroutine(RunCurrentCommand());
            }

            else
            {
                if(foodDecrease){
                    LoseFood(currentCommand.steps*stepMultiplier);
                }
                foodText.text = "Food: " + food;
                commandIndex++;
                currentSteps = 0;
                moving = true;
            }
        }
        else
        {
            FinishedRunningCommand();
            running = false;
        }

    }
}

1 reference
public void SetCommands(Command[] commands)
{
    this.commands = commands;
    commandIndex = 0;
    running = true;
}
```

Gambar 4.30 Player.cs mengolah variabel command yang telah diterima

```
public void Sent()
{
    List<string> simplifiedCommands = lines.Select(x => SimplifyStringToEssentials(x)).ToList(); //extract command and convert it to String then store it in simplifiedCommands;
    Command[] commands = Command.CommandsFromLines(simplifiedCommands); //send data to command script and call ComandsFromLines Function
    Player player = FindObjectOfType<Player>();
    player.SetCommands(commands);
    GameManager gameManager = FindObjectOfType<GameManager>();
}
```

Gambar 4.31 Fungsi Sent() pada skrip Console.cs

variabel *command* yang telah disederhanakan akan dikirimkan melalui fungsi *Sent()* yang ada pada skrip *Console.cs* (Gambar 4.31). variabel ini akan diterima dan dimasukan pada variabel *local* skrip *Player.cs* dan mengubah *boolean running*.

Pada Gambar 4.30 ketika *boolean running* kembali sebagai *true* maka *command* yang ada pada *array* akan diubah satu persatu menjadi fungsi untuk menggerakan pemain, melakukan perubahan pada variabel *Food Point*.

```
1 reference
IEnumerator RunCurrentCommand()
{
    if (currentSteps < currentCommand.steps)
    {
        switch (currentCommand.commandType)
        {
            case Command.CommandType.Up:
                direction = new Vector3(0, 1, 0);
                MovePlayer();
                break;
            case Command.CommandType.Down:
                direction = new Vector3(0, -1, 0);
                MovePlayer();
                break;
            case Command.CommandType.Left:
                direction = new Vector3(-1, 0, 0);
                MovePlayer();
                transform.localRotation = Quaternion.Euler(0, 180, 0);
                break;
            case Command.CommandType.Right:
                direction = new Vector3(1, 0, 0);
                MovePlayer();
                transform.localRotation = Quaternion.Euler(0, 0, 0);
                break;

            case Command.CommandType.UpAttack:
                direction = new Vector3(0, 1, 0);
                Attack();
                break;
            case Command.CommandType.LeftAttack:
                direction = new Vector3(-1, 0, 0);
                transform.localRotation = Quaternion.Euler(0, 180, 0);
                Attack();
                break;
            case Command.CommandType.DownAttack:
                direction = new Vector3(0, -1, 0);
                Attack();
                break;
            case Command.CommandType.RightAttack:
                direction = new Vector3(1, 0, 0);
                transform.localRotation = Quaternion.Euler(0, 0, 0);
                Attack();
                break;
            case Command.CommandType.Ignore:
                currentSteps++;
                break;
        }
    }
    else if(currentSteps == currentCommand.steps)
    {
        moving = false;
        //Debug.Log(currentCommand.steps);
    }
    yield return null;
}
```

Gambar 4.32 Fungsi RunCurrentCommand()

Gambar 4.32 merupakan sebuah fungsi yang digunakan untuk mengecek *CommandType* pada *currentCommand* dan mengubahnya menjadi fungsi untuk menggerakan karakter pemain.

```
1 reference
private void HandleControlInput() //directional system di console, arrow, enter, backspace
{
    bool shift = Input.GetKey(KeyCode.LeftShift) || Input.GetKey(KeyCode.RightShift);

    //New Line
    if (Input.GetKeyDown(KeyCode.Return))
    {
        AudioM.PlayKeyPress(true);
        bool specialCommandEntered = OnTextEntered(selectedLineIndex); //check apakah ini line special command
        if (specialCommandEntered)
        {
            caretCharIndex = lines[selectedLineIndex].Length;
        }
        else
        {
            lastKeyTime = Time.time;
            if(lines.Count < maxNumLinesOnScreen)
            {
                if(selectedLineIndex < lines.Count - 1)
                {
                    lines.Insert(selectedLineIndex + 1, "");
                }
                else
                {
                    lines.Add("");
                }
                selectedLineIndex++;
                caretCharIndex = 0;
            }
            else
            {
                selectedLineIndex++;
                selectedLineIndex = Mathf.Clamp(selectedLineIndex, 0, maxNumLinesOnScreen - 1);
                caretCharIndex = lines[selectedLineIndex].Length;
            }
        }
    }
    //Backspace
    if (CustomInput.instance.GetKeyPress(KeyCode.Backspace))
    {
        lastKeyTime = Time.time;
        if(lines[selectedLineIndex].Length == 0)
        {
            if(lines.Count != 1)
            {
                lines.RemoveAt(selectedLineIndex);
                if(selectedLineIndex > 0)
                {
                    selectedLineIndex--;
                    caretCharIndex = lines[selectedLineIndex].Length;
                    AudioM.PlayKeyPress(false);
                }
            }
        }
```

Gambar 4.33 Fungsi untuk menginput perintah

```
        else
        {
            if(caretCharIndex > 0)
            {
                AudioM.PlayKeyPress(false);
                caretCharIndex--;
                lines[selectedLineIndex] = lines[selectedLineIndex].Remove(caretCharIndex, 1);
            }
        }
    }

    int lineIndexOld = selectedLineIndex;
    int charIndOld = caretCharIndex;
    //Arrow keys
    if (CustomInput.instance.GetKeyPress(KeyCode.UpArrow))
    {
        lastKeyTime = Time.time;
        if (shift)
        {
            selectedLineIndex = 0;
        }
        else
        {
            selectedLineIndex = Mathf.Clamp(selectedLineIndex - 1, 0, int.MaxValue);
        }
        caretCharIndex = lines[selectedLineIndex].Length;
    }
    if (CustomInput.instance.GetKeyPress(KeyCode.DownArrow))
    {
        lastKeyTime = Time.time;
        if (shift && lines.Count>0)
        {
            selectedLineIndex = lines.Count - 1;
        }
        else
        {
            selectedLineIndex = Mathf.Clamp(selectedLineIndex + 1, 0, lines.Count - 1);
        }
        caretCharIndex = lines[selectedLineIndex].Length;
    }
    if (CustomInput.instance.GetKeyPress(KeyCode.LeftArrow))
    {
        lastKeyTime = Time.time;
        if (shift)
        {
            caretCharIndex = 0;
        }
        else
        {
            caretCharIndex = Mathf.Clamp(caretCharIndex - 1, 0, int.MaxValue);
        }
    }
    if (CustomInput.instance.GetKeyPress(KeyCode.RightArrow))
    {
        lastKeyTime = Time.time;
        if (shift)
        {
            caretCharIndex = 0;
        }
        else
```

Gambar 4.33 Fungsi untuk menginput perintah (lanjutan)

```
            else
            {
                caretCharIndex = Mathf.Clamp(caretCharIndex + 1, 0, lines[selectedLineIndex].Length);
            }
        }

        if(selectedLineIndex != lineIndexOld || caretCharIndex != charIndOld)
        {
            AudioM.PlayKeyPress(false);
        }
    }
```

Gambar 4.33 Fungsi untuk menginput perintah (lanjutan)

Pada fungsi *HandleControlInput* yang terdapat pada Gambar 4.33 memiliki fungsi sebagai pengontrol *input*. *Input* seperti *keystroke*, spasi, *backspace*, *enter key*, dan *arrow keys*, akan diatur kegunaannya pada fungsi ini. *HandleControlInput* ini dipanggil pada fungsi *Update* yang ada pada Gambar 4.34.

```
void Update()
{
    //text input
    bool enteredString = false;
    string input = Input.inputString.ToLower();

    if(!TutorialMode.tutorialMode){

        foreach(char c in input)
        {
            if(lines[selectedLineIndex].Length >= charLimit)
            {
                break;
            }
            if (legalChars.Contains(c.ToString().ToLower()))
            {
                enteredString = true;
                if(caretCharIndex < lines[selectedLineIndex].Length)
                {
                    lines[selectedLineIndex] = lines[selectedLineIndex].Insert(caretCharIndex, c.ToString());
                }
                else
                {
                    lines[selectedLineIndex] += c;
                }
                caretCharIndex++;
                lastKeyTime = Time.time;
            }
        }

    }
    helpSplash.gameObject.SetActive(helpMenuOpen);

    if (enteredString)
    {
        AudioM.PlayKeyPress(false);
    }
    HandleControlInput();
    UpdateDisplay();
}
```

Gambar 4.34 Fungsi Update pada skrip Console.cs

Fungsi *Update* merupakan fungsi dasar *Unity* yang akan terus berjalan sepanjang permainan dimainkan. Pada skrip *Console.cs* fungsi *Update* digunakan

untuk mengecek apakah pemain sedang melakukan *input* atau tidak dan juga mengubah karakter kapital menjadi non kapital, fungsi pada Gambar 4.33 juga dipanggil pada fungsi *Update* ini juga.

```
1 reference
void UpdateDisplay()
{
    if(selectedLineIndex < firstDisplayedLineIndex)
    {
        firstDisplayedLineIndex = selectedLineIndex;
    }
    if(selectedLineIndex >= firstDisplayedLineIndex + maxNumLinesOnScreen)
    {
        firstDisplayedLineIndex = selectedLineIndex - maxNumLinesOnScreen + 1;
    }

    for(int i = 0; i < textFields.Length; i++)
    {
        textFields[i].text = "";
    }

    for(int i = 0; i < textFields.Length; i++)
    {
        if(firstDisplayedLineIndex + i > lines.Count - 1)
        {
            break;
        }
        textFields[i].text = lines[firstDisplayedLineIndex + i];
    }

    //Draw Caret
    bool caretVisible = false;
    if(Time.time - lastKeyTime > blinkDelay)
    {
        caretVisible = (int)((Time.time - (lastKeyTime + blinkDelay)) / blinkTime) % 2 == 0;
    }
    else
    {
        caretVisible = true;
    }

    if (caretVisible)
    {
        caret.enabled = true;
        Text selectedField = textFields[selectedLineIndex - firstDisplayedLineIndex];
        selectedField.font.RequestCharactersInTexture(selectedField.text, selectedField.fontSize, selectedField.fontStyle);

        float caretOffsetX = 0;

        for(int i = 0; i < caretCharIndex; i++)
        {
            CharacterInfo info;
            selectedField.font.GetCharacterInfo(selectedField.text[i], out info, selectedField.fontSize, selectedField.fontStyle);
            caretOffsetX += info.advance;
        }

        caret.rectTransform.position = selectedField.rectTransform.position;
        caret.rectTransform.localPosition += Vector3.right * (caretOffsetX + caret.rectTransform.rect.width / 2f);
    }
    else
    {
        caret.enabled = false;
    }
}
```

Gambar 4.35 Fungsi UpdateDisplay()

Pada fungsi *Update* pada Gambar 4.34 terdapat sebuah fungsi yang juga dipanggil bernama fungsi *UpdateDisplay* yang terdapat pada Gambar 4.35. Fungsi ini digunakan untuk memberikan *feedback* kepada pemain ketika pemain melakukan *input*, *feedback* ini berupa memunculkan huruf yang pemain ketikan, *caret* sebagai penunjuk baris dan kolom muncul dan mengikuti input dan juga *highlight* ketika pemain mengeksekusi perintah.

- *Character Movement*

Pergerakan karakter pemain diatur melalui skrip *Player.cs* dan dipanggil berdasarkan jenis perintah yang dimasukkan.

```
4 references
void MovePlayer()
{
    if (Vector3.Distance(transform.position, movePoint.position) <= .05f)
    {
        RaycastHit2D hit = Physics2D.Raycast(movePoint.position, direction, 1, blockingLayer);
        if (!hit)
        {
            movePoint.position += direction;
            currentSteps++;
            foodDecrease = true;
        }
        else
        {
            currentSteps++;
            Debug.Log("command ignored");
        }

    }
}
4 references
void Attack()
{
    if (Vector3.Distance(transform.position, movePoint.position) <= .05f)
    {
        RaycastHit2D hit = Physics2D.Raycast(movePoint.position, direction, 1, blockingLayer);
        if (!hit)
        {
            currentSteps++;
            Debug.Log("I Hit nothing");
        }
        else if(hit && hit.collider.gameObject.tag == "Wall"){
            currentSteps++;
            Debug.Log("You hit hard wall");
        }
        else
        {
            currentSteps++;
            Destroy(hit.collider.gameObject);
            Debug.Log("I hit wall");
            foodDecrease = true;
        }
        anim.SetTrigger("Attack");
    }
}
```

Gambar 4.36 Fungsi untuk menggerakan karakter pemain.

Setelah fungsi *RunCommand* pada Gambar 4.32 berjalan *currentCommand* akan di cek dan dicocokkan dengan *enum* yang ada pada skrip *Command.cs* dan berdasarkan *enum* yang ada variabel *vector3* bernama *direction* ditetapkan yang kemudian variabel ini akan diterima ketika fungsi *MovePlayer()* atau *Attack()*

dipanggil. Gambar 4.36 merupakan fungsi *MovePlayer()* dan *Attack()* kedua fungsi ini membutuhkan *direction* yang dimana variabel ini didapat dari fungsi sebelumnya.

Pada fungsi *MovePlayer()* sebuah raycast akan ditembakan ke arah *direction* yang telah diterima sebelumnya, *raycast* ini berfungsi untuk mengecek apakah *direction* yang dituju terdapat penghalang atau tidak, ketika *raycast* ini mendeteksi sebuah halangan maka karakter pemain tidak akan bergerak ke arah *direction* akan tetapi jika tidak terdapat halangan karakter pemain akan bergerak berdasarkan *direction* yang telah ditetapkan.

Fungsi *Attack()* memiliki sebuah raycast yang ditembakan pula sama seperti fungsi *MovePlayer()* akan tetapi ketika *raycast* mendeteksi adanya halangan, penghalang ini akan di *destroy*.

IV. 6 Pengujian Statistik

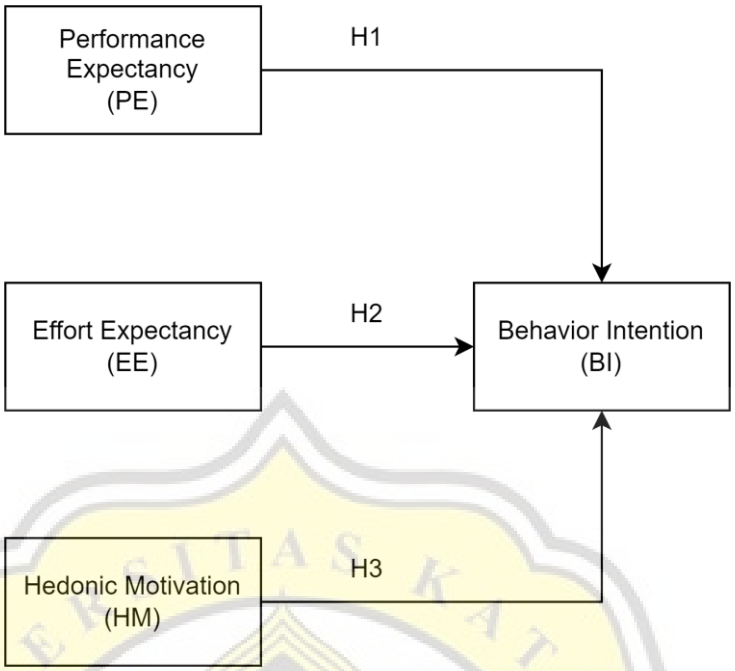
Setelah menyelesaikan pengembangan *game* "Take Me Home" maka peneliti harus menentukan hipotesa yang berguna untuk memprediksi hasil dari penelitian dalam mengenalkan teknik pengembangan *computational thinking.* Berikut rancangan hipotesa yang telah disusun:

H1: Pengenalan teknik pembelajaran *computational thinking* melalui contoh *game* "Take Me Home" menarik dan mengedukasi

H2: Pengenalan teknik pembelajaran *computational thinking* melalui contoh *game* "Take Me Home" dapat dipahami.

H3. Contoh game "Take Me Home" memiliki permainan yang menarik dan menantang untuk dimainkan.

Dari rancangan hipotesa yang telah dibuat Gambar 4.37 menunjukan hipotesa korelasi tiap variabel, korelasi ini berupa hubungan antara *Performance Expectancy* dan *Behavior Intention*, hubungan antara *Effort Expectancy* dan *Behavior Intention* dan terakhir hubungan antara *Hedonic Motivation* dan *Behavior Intention*.



Gambar 4.37 Hipotesa Relasi Antar variabel

IV.6.1 Rancangan Pertanyaan Kuesioner

Untuk mendapatkan hasil maksimal dari responden peneliti telah merancang daftar pertanyaan yang ditetapkan berdasarkan hipotesa yang telah disusun. berikut merupakan daftar pertanyaan yang akan menjadi tolak ukur dalam penelitian.

- *Performance Expectancy*

  PE1. Ketika memainkan *game* "Take Me Home" berapa banyak level yang berhasil anda lewati.

  PE2. Dengan memainkan *game* "Take Me Home" dapat membantu saya dalam melihat dan menganalisa sebuah masalah.

  PE.3 Dengan memainkan *game* "Take Me Home" saya dapat membentuk strategi untuk menyelesaikan masalah.

- *Effort Expectancy*

  EE1. Memahami cara bermain *game* "Take Me Home" mudah bagi saya.

  EE2. Menurut saya, instruksi bermain di game "Take Me Home" sangat jelas dan mudah dipahami.

EE3. Konsep permainan dalam game "Take Me Home" mudah untuk dipahami.

EE4. Saya sangat cepat memahami cara bermain *game* "Take Me Home".

- *Hedonic Motivation*

  HM1. Menurut saya, permainan yang ada dalam *game* "Take Me Home" menarik.

  HM2. Menurut saya, permainan "Take Me Home" sangat menantang.

  HM3. Memainkan *game* "Take Me Home" dapat membuat saya merasa senang.

- *Behavioral Intention*

  BI1. Saya berencana untuk tetap memainkan *game* "Take Me Home" di kemudian hari.

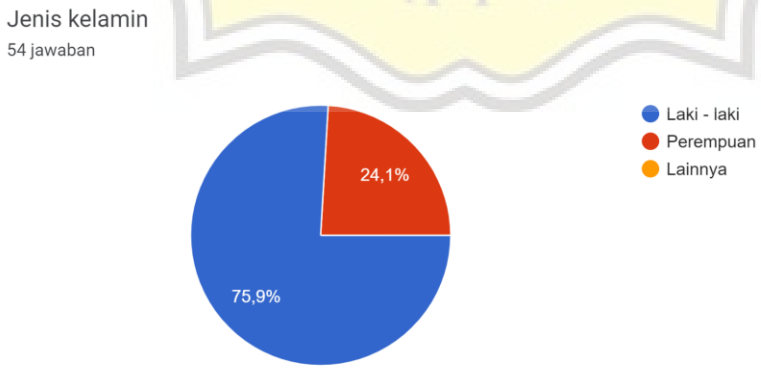
  BI2. Saya akan terus mencoba memainkan *game* "Take Me Home" di kehidupan saya sehari - hari

  BI3. Saya berencana memainkan *game* "Take Me Home" lebih sering.

IV.6.2 Data Responden

1. Jenis Kelamin

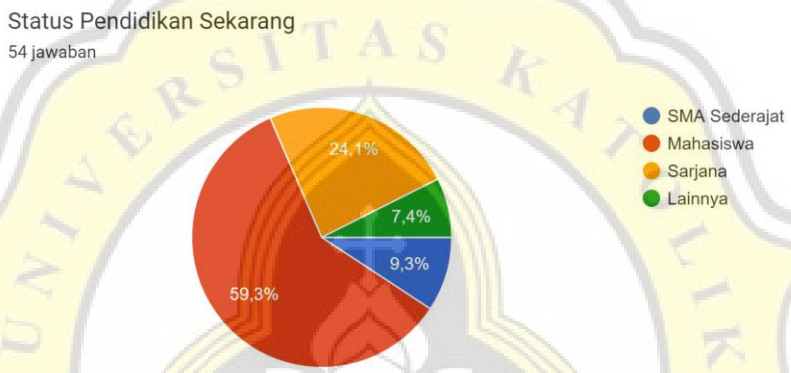
   Pada Gambar 4.38 dapat dilihat bahwa dari 54 total responden, jenis kelamin terbanyak adalah laki - laki dengan nilai persentase 75,9% atau sebanyak 41 responden, sedangkan jenis kelamin perempuan memiliki nilai persentase 24,1% atau sebanyak 13 responden.



Gambar 4.38 Persentase Jenis Kelamin

2. Status Pendidikan Sekarang

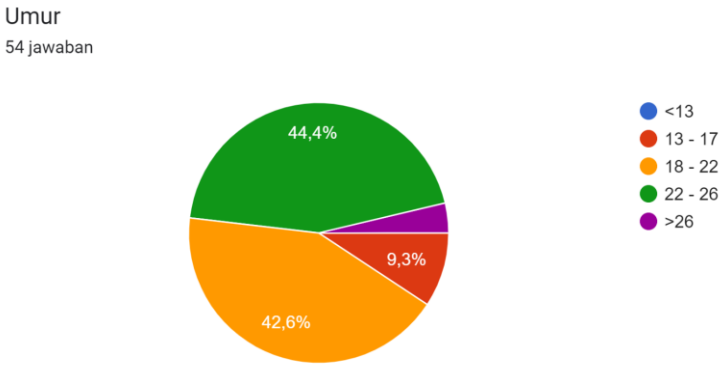
Dapat dilihat pada Gambar 4.39 mayoritas responden memiliki status pendidikan mahasiswa dengan nilai persentase sebanyak 59.3% atau 32 responden dari total 54 responden, selanjutnya 24.1% atau 13 responden dari total 54 responden memiliki status pendidikan sarjana, selanjutnya terdapat 9.3% atau 5 responden yang memiliki status pendidikan masih menempuh masa SMA sederajat, dan terakhir terdapat 7.4% atau 4 responden yang memiliki status pendidikan di luar dari pilihan yang telah ditentukan.



Gambar 4.39 Persentase status pendidikan sekarang

3. Usia

Pada gambar 4.40 menunjukan dari 54 responden, mayoritas responden memiliki rentang usia 22-26 tahun dengan nilai persentase 44.4% atau berjumlah 24 responden tiap kategorinya, lalu untuk responden berumur 18-22 tahun terdapat 42,6% atau sebanyak 23 responden kemudian terdapat 9.3% atau 5 responden memiliki rentang usia 13-17 tahun, dan terakhir terdapat 3.7% atau sebanyak 2 responden memiliki rentang usia lebih dari 26 tahun.



Gambar 4.40 Persentase Usia

IV.6.3 Statistik Deskriptif

Statistik Deskriptif dapat dilihat pada Tabel 4.1, tabel ini berisikan tentang data keseluruhan responden, dengan total 54 responden dan dengan ketentuan pada tabel Gender angka 1 adalah laki - laki dan angka 2 adalah perempuan. Pada tabel Education angka 1, responden sedang menempuh pendidikan SMA Sederajat, angka 2, responden merupakan seorang Mahasiswa, angka 3, responden kini sudah memiliki status pendidikan Sarjana, dan angka 4, responden memiliki pendidikan lainnya. Pada tabel umur angka 1 menunjukan kisaran usia dibawah 13 tahun, lalu angka 2 menunjukan kisaran usia 13-17 tahun, angka 3 menunjukan kisaran usia 18-22 tahun, angka 3 menunjukan kisaran usia 22-26, dan angka 4 menunjukan responden yang memiliki usia diatas 26 tahun. Untuk keterangan tabel gender memiliki rata-rata 1.24 dan nilai tengah 1, lalu untuk keterangan tabel edukasi memiliki nilai rata-rata 2.30 dan memiliki nilai tengah 2, sedangkan untuk keterangan tabel umur rata-rata nilai yang dimiliki bernilai 2.44 dan memiliki nilai tengah 2.50.

Tabel 4.1 Statistik Deskriptif

Statistics

		Gender	Edu	Age
N	Valid	54	54	54
	Missing	0	0	0
Mean		1.24	2.30	2.44
Median		1.00	2.00	2.50
Mode		1	2	3
Sum		67	124	132

IV.6.4 Uji Validasi

Pengujian validitas diperlukan untuk mengetahui apakah instrumen yang akan diujikan merupakan data valid atau tidak valid. Data yang digunakan merupakan data yang didapatkan dari kuesioner yang telah disebarkan kepada responden[27]. Pada game "Take Me Home" variabel yang akan diujikan adalah variabel PE, EE, HM, dan BI. Pada hasil uji validitas yang terdapat pada Tabel 4.2 seluruh variabel valid dikarenakan memiliki pola yang serupa dan memiliki nilai diatas 0.4..

Tabel 4.2 Uji Validasi

Rotated Component Matrix[a]

	Component		
	1	2	3
PE1	.034	.836	.102
PE2	.219	.759	.045
PE3	.312	.601	.202
EE1	.089	-.046	.824
EE2	-.001	.471	.616
EE3	.141	.570	.564
EE4	.365	.145	.587
HM1	.445	.505	.302
HM2	.720	.099	.258
HM3	.797	.084	.316
BI1	.801	.362	.134
BI2	.825	.116	.053
BI3	.817	.356	.089

Extraction Method: Principal Component Analysis.
Rotation Method: Equamax with Kaiser Normalization.

a. Rotation converged in 6 iterations.

IV.6.5 Uji Reliabilitas

Uji reliabilitas digunakan untuk melihat tingkat konsistensi dari sebuah data, tingkat konsistensi ini dapat dilihat dari hasil yang didapatkan akan tetap sama ketika responden melakukan kuesioner ulang [28]. Pada tahap ini uji validitas diperlukan untuk melihat tingkat konsistensi sebuah data, penggunaan uji validitas dimaksudkan untuk melihat apa saja variabel yang valid untuk diuji reliabilitasnya. Dari hasil uji validitas pada Tabel 4.3 semua variabel merupakan variabel valid dengan demikian berikut merupakan hasil dari uji reliabilitas tiap variabel.

Tabel 4.4 variabel PE memiliki hasil 0.726 pada uji reliabilitas.

Tabel 4.4 Reliabilitas PE

Reliability Statistics

Cronbach's Alpha	Cronbach's Alpha Based on Standardized Items	N of Items
.727	.726	3

Tabel 4.5 Reliabilitas EE

Reliability Statistics

Cronbach's Alpha	Cronbach's Alpha Based on Standardized Items	N of Items
.712	.716	4

Tabel 4.6 Reliabilitas HM

Reliability Statistics

Cronbach's Alpha	Cronbach's Alpha Based on Standardized Items	N of Items
.742	.745	3

Tabel 4.7 Reliabilitas BI

Reliability Statistics

Cronbach's Alpha	Cronbach's Alpha Based on Standardized Items	N of Items
.884	.884	3

Untuk melihat nilai dari uji validitas diperlukan keterangan mengenai standar rentang nilai yang digunakan untuk melakukan penilaian suatu variabel. Tabel 4.8 menunjukan rentang nilai standar untuk uji reliabilitas.

Tabel 4.8 Rentang Nilai Uji Reliabilitas

Cronbach's Alpha	**Internal Consistency**
$\alpha \geq 0.9$	Excellent
$0.9 > \alpha \geq 0.8$	Good
$0.8 > \alpha \geq 0.7$	Acceptable
$0.7 > \alpha \geq 0.6$	Questionable
$0.6 > \alpha \geq 0.5$	Poor
$0.5 > \alpha$	Unacceptable

Pada Tabel 4.9 dapat disimpulkan bahwa variabel PE,EE dan HM memiliki hasil *Acceptable*, lalu variabel BI memiliki hasil *Good.*

Tabel 4.9 Hasil Uji Reliabilitas

Variabel	**Cronbach's Alpha**	**Internal Consistency**
PE	0.726	Acceptable
EE	0.716	Acceptable
HM	0.745	Acceptable
BI	0.884	Good

IV.6.6 Uji Korelasi

Dari hasil yang ditampilkan pada Tabel 4.10 dapat dilihat bahwa nilai variabel yang memiliki hubungan bagus memiliki tanda bintang sebagai petunjuk bahwa variabel yang ditunjuk memiliki korelasi yang kuat. Hasil korelasi antara rata - rata variabel PE, EE, HM, dan BI yang telah dirubah menjadi RPE, REE, RHM, dan RBI mendapatkan kesimpulan bahwa:

H1 terbukti variabel PE dan BI memiliki korelasi yang baik dengan nilai korelasi diatas 0.4 dan ditandai

H2 terbukti variabel EEdan BI memiliki korelasi yang baik dengan nilai korelasi diatas 0.4 dan ditandai

H3 terbukti variabel HM dan BI memiliki korelasi yang baik dengan nilai korelasi diatas 0.4 dan ditandai

Tabel 4.10 Hasil Uji Validasi

Correlations

		RPE	REE	RHM	RBI
RPE	Pearson Correlation	1	.498**	.499**	.464**
	Sig. (2-tailed)		.000	.000	.000
	N	54	54	54	54
REE	Pearson Correlation	.498**	1	.542**	.422**
	Sig. (2-tailed)	.000		.000	.001
	N	54	54	54	54
RHM	Pearson Correlation	.499**	.542**	1	.768**
	Sig. (2-tailed)	.000	.000		.000
	N	54	54	54	54
RBI	Pearson Correlation	.464**	.422**	.768**	1
	Sig. (2-tailed)	.000	.001	.000	
	N	54	54	54	54

**. Correlation is significant at the 0.01 level (2-tailed).

Tabel yang dihasilkan dari uji korelasi menunjukkan bahwa terdapat hubungan yang signifikan antara variable "PE" dan variable "BI". Berdasarkan ini, dapat disimpulkan bahwa game "Take Me Home" terbukti efektif dalam mengajarkan dan meningkatkan kemampuan Computational Thinking seseorang. Jadi, dapat disimpulkan bahwa game tersebut memiliki kemampuan yang efektif dalam meningkatkan kemampuan Computational Thinking seseorang.