BAB 3
METODOLOGI PENELITIAN

3.1 Identifikasi Masalah

Pada tahap identifikasi masalah, dilakukan observasi terhadap tren-tren penggunaan algoritma sebagai penyampaian pesan pada beberapa *game* salah satu observasi dilakukan pada *thread* yang diunggah pada forum-forum. Hasil observasi menunjukkan bahwa penggunaan teks yang di enkripsi sering digunakan dalam *game* dalam bentuk *easter egg* atau sebagai aksara bahasa buatan dalam sebuah *game* fantasi. Selain itu dilakukan observasi terhadap hasil survey mengenai tingkat literasi dan juga dilakukan observasi terhadap penggunaan algoritma *Caesar Cipher* sebagai dasar pembelajaran kriptografi. Berdasarkan hal hal tersebut ditemukan sebuah masalah untuk diteliti yang dilakukan dengan pembuatan *game* sebagai media belajar *Caesar Cipher*.

3.2 Studi Literatur

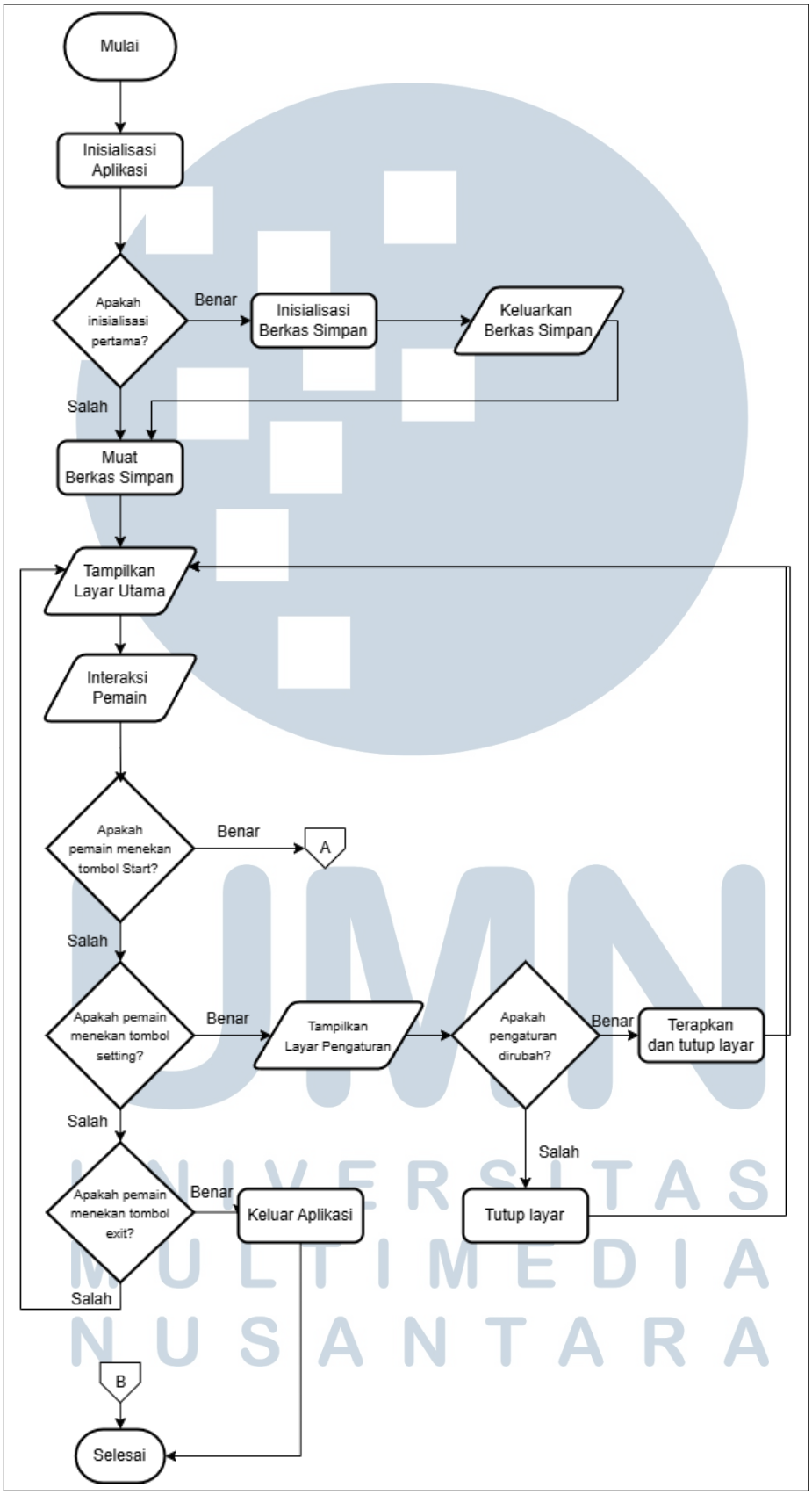
Setelah menemukan topik masalah untuk diteliti, dilakukan Studi Literatur, pada tahap ini dilakukan pembelajaran teori mengenai algoritma terkait seperti *Caesar Cipher*, serta teori-teori yang berhubungan dengan penelitian seperti Kriptografi, *Random Number Generator*, *Achievement*, *USE Questionnaire* dan *Likert Scale*.

3.3 Merancang *Game*

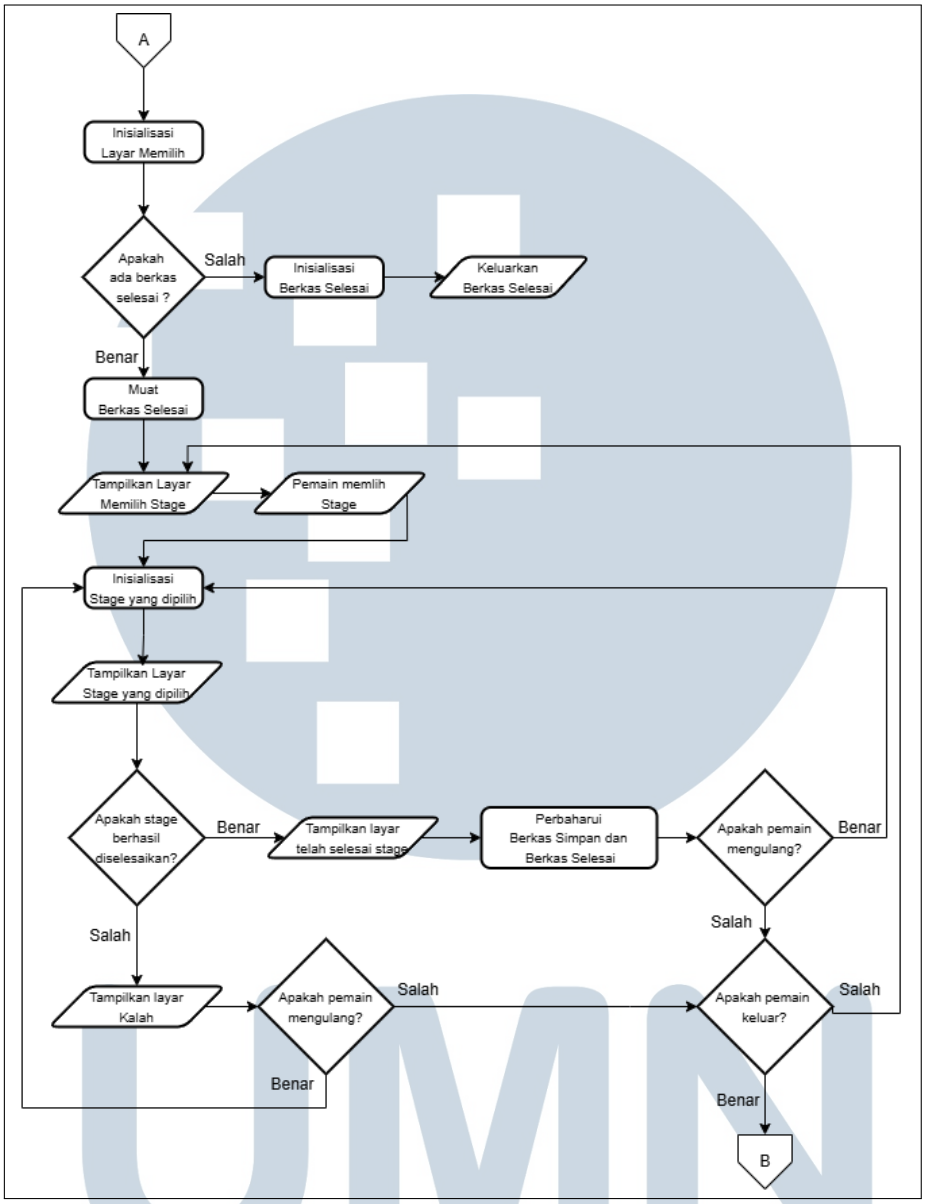
Setelah melakukan studi literatur, dilakukan peracangan dan pembangunan *game*, tahap perancangan terdiri dari perancangan alur aplikasi *game* , perancangan sistem pengambilan pertanyaan dan perancangan *stage* sebagai area permainan.

3.3.1 Perancangan Alur Aplikasi *Game*

Pada proyek ini, Aplikasi *game* yang dirancang memiliki alur seperti yang diilustrasikan pada *flowchart* yang ada pada Gambar 3.1 dan Gambar 3.2.



Gambar 3.1. *Flowchart* alur aplikasi *game*

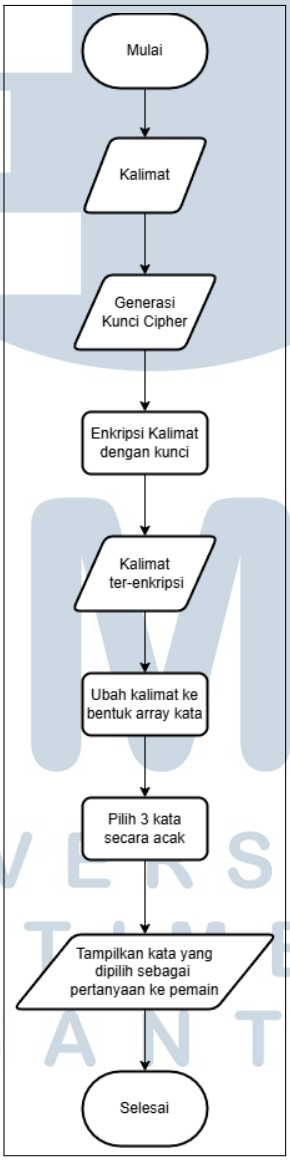


Gambar 3.2. *Flowchart* alur aplikasi *game*

Pada Gambar 3.1, Terdapat fitur penyimpanan secara otomatis, dimana proses pembuatan *save file* dilakukan pada saat aplikasi *game* dijalankan untuk pertama kali, dan akan di perbaharui setiap kali terdapat data *stage clear data* baru setelah menyelesaikan *stage* pada *game*. Pada Gambar 3.2 dapat dilihat proses initialisasi *stage* dimana dilakukan proses enkripsi dengan *Caesar Chipher* pada kalimat yang akan menjadi pertanyaan di *stage* yang dipilih. Illustrasi proses pemilihan kata dari kalimat yang telah di enkripsi dapat dilihat pada Gambar 3.4

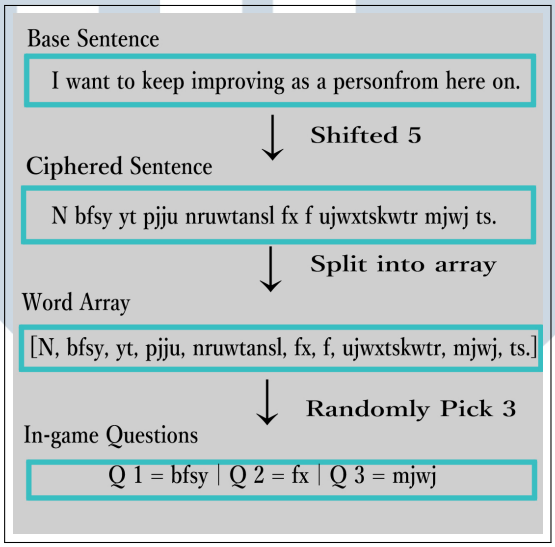
3.3.2 Perancangan Pengambilan Pertanyaan

Pada tahap ini, desain *logic game* dibuat, *Caesar Cipher* diimplementasikan sebagai pertanyaan disetiap *stage*, dimana tiap *stage* akan terdiri dari tiga *level*. Kata yang ditampilkan akan diambil secara acak dari total kata-kata yang digunakan untuk membentuk suatu paragraf yang berkaitan dengan latar belakang karakter, kata yang ditampilkan juga diacak jumlah pergeseran hurufnya. *Flowchart* konsep pemilihan kata yang akan menjadi pertanyaan dapat dilihat pada Gambar 3.3 dan illustrasi contoh pemilihan kata untuk dijadikan pertanyaan dapat dilihat pada Gambar 3.4



Gambar 3.3. *Flowchart* pemilihan kata

Pada Gambar 3.3 dapat dilihat pengambilan kata untuk pertanyaan berawal dari melakukan enkripsi pada satu kalimat utuh, yang setelah di enkripsi, pada proses enkripsi kalimat ini, kalimat akan dienkripsi menggunakan algoritma *Caesar Cipher* setelah itu kalimat yang telah di enkripsi akan dirubah kebentuk *array* agar dapat dilakukan pengambilan kata secara acak, setelah itu sebelum ditampilkan ke pemain, kata yang telah dipilih akan diacak ulang untuk menentukan urutan ditampilkan ke pemain. Illustrasi pengaplikasian konsep pemilihan kata dapat dilihat pada 3.4.

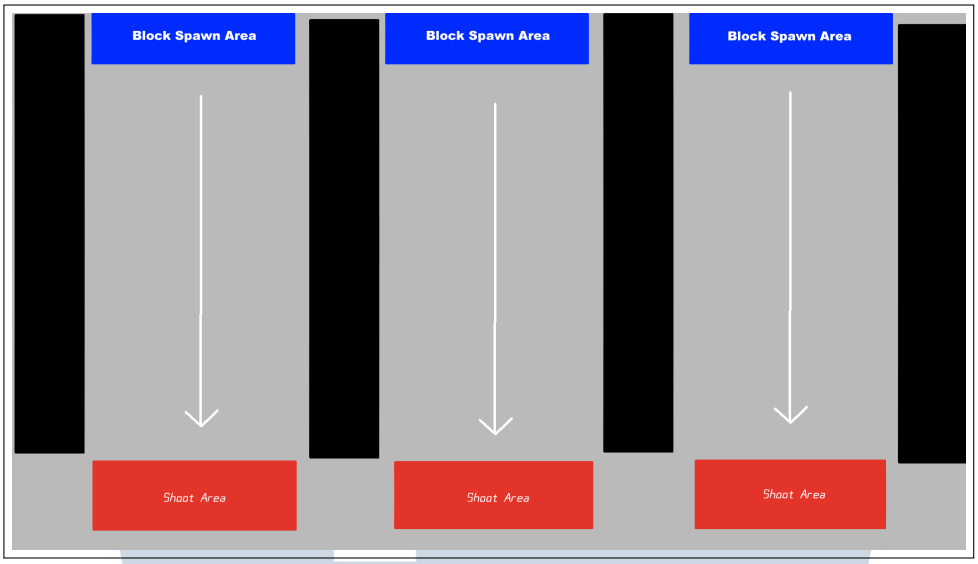


Gambar 3.4. Illustrasi pemilihan kata

Terdapat tiga bagian kode inti dari *Caesar Cipher* yang diimplementasi pada tahap ini, yaitu enkripsi dan dekripsi serta kode untuk melakukan pergeseran pada saat enkripsi. Kode untuk melakukan pergeseran huruf dapat dilihat pada Kode 3.1.

3.3.3 Perancangan Stage

Pada tiap *level*, tingkat kesulitan ditentukan dari kecepatan block mendatangi *player*, visualisasi dapat terlihat pada Gambar 3.5. Dimana setiap *player* menyelesaikan satu *level*, *player* dapat berjalan ke *shoot area* pada *level* berikutnya.



Gambar 3.5. Illustrasi tingkat kesulitan

3.4 Membangun *Game*

Setelah melakukan perancangan, dilakukan pembangunan aplikasi *game* yang terdiri dari tiga tahap, yaitu, Implementasi *Caesar Cipher*, Implementasi *Level* dan Implementasi Achievement

3.4.1 Implementasi *Caesar Cipher*

```
1     private static char Chiper(char ch, float key)
2     {
3       if (!char.IsLetter(ch))
4       {
5           return ch;
6       }
7       char offset = char.IsUpper(ch) ? 'A' : 'a';
8       return (char)((((ch + key) - offset) % 26) + offset);
9     }
```

Kode 3.1: Implementasi cipher pada game

Pergeseran akan dilakukan terhadap setiap input berupa huruf yang diterima oleh modul program sebanyak jumlah pergeseran yang telah ditentukan, pada modul pergeseran akan dilakukan pemeriksaan apakah input yang diterima adalah huruf kapital atau tidak. Modul pergeseran akan dipanggil dari modul Enkripsi yang dapat dilihat pada Kode 3.2. Pada modul enkripsi, modul akan menerima input

berupa kalimat dan kunci untuk melakukan pergeseran. Kalimat yang diterima akan dibagi menjadi perhuruf, setelah terbagi pergeseran akan dilakukan perhuruf, dan huruf yang sudah digeser akan disatukan pada satu variabel *string* yang menjadi output. Kode dekripsi dapat dilihat pada Kode 3.3 proses dekripsi dilakukan dengan cara menjalankan modul enkripsi dengan kunci yang sudah dikurangi 26 untuk mengembalikan input ke bentuk asal.

```
1    public static string EncodeCaesar(string inp, float key)
2    {
3        string output = string.Empty;
4        foreach (char ch in inp)
5        {
6            output += Chiper(ch, key);
7        }
8        return output;
9    }
```

Kode 3.2: Implementasi Encode Caesar Chiper pada game

```
1    public string DecodeCaesar(string inp, float key)
2    {
3        return EncodeCaesar(inp, 26 - key);
4    }
```

Kode 3.3: Implementasi Decode Caesar Chiper pada game

Untuk menghindari repetisi dan kecurangan dalam *game*, pada setiap kali *level* di-initialisasi kunci untuk melakukan pergeseran kata akan dipilih secara acak, seperti yang terlihat pada Kode 3.4.

```
1    float chiperkey = UnityEngine.Random.Range(1, 26);
```

Kode 3.4: Implementasi Decode Caesar Chiper pada game

3.4.2 Implementasi Level

Pada tahap ini, desain *level* dibuat, selain itu tingkat kesulitan juga diatur pada proses ini. Tingkat kesulitan akan diimplementasikan dalam bentuk kecepatan blok jawaban saat mendekati *player*.

Setiap *stage* pada *game* ini terdiri dari tiga *level*, dan tiap *level* berisikan pertanyaan yang berdasar pada satu paragraf yang sudah ditentukan. Seperti yang terlihat pada Kode 3.5, pada saat *level* di-*load*, program akan mengambil data kalimat dari *PlayerPrefs* untuk di proses, kalimat akan digeser sejumlah kunci yang

sudah ditentukan, kemudian kalimat yang telah digeser akan dipecah ke bentuk per-kata, setelah itu dari kata-kata tersebut akan dipilih 3 kata secara acak untuk menjadi pertanyaan pada *stage* saat *run-time* ini. Proses ini dilakukan setiap *level* di-*load* oleh *player*. Untuk memudahkan *player*, kunci yang digunakan untuk menggeser kalimat akan ditampilkan dalam bentuk *hint*.

```
1    void LoadInitialData()
2    {
3        currStageSentence = PlayerPrefs.GetString("StageSentence")
    ;
4    }
5
6    void InitializeCurrentLevel()
7    {
8        float chiperkey = UnityEngine.Random.Range(1, 26);
9        checkkey = chiperkey;
10       string chiperedSentence = EncodeCaesar(currStageSentence,
    chiperkey);
11       string[] checksplit = chiperedSentence.Split(separator);
12       int questionkey1 = UnityEngine.Random.Range(0, checksplit.
    Length);
13       int questionkey2 = UnityEngine.Random.Range(0, checksplit.
    Length);
14       int questionkey3 = UnityEngine.Random.Range(0, checksplit.
    Length);
15       pickedquestion1 = checksplit[questionkey1].ToString();
16       question.text = pickedquestion1;
17       pickedquestion2 = checksplit[questionkey2].ToString();
18       question2.text = pickedquestion2;
19       pickedquestion3 = checksplit[questionkey3].ToString();
20       question3.text = pickedquestion3;
21       hint1.text = checkkey.ToString();
22       hint2.text = checkkey.ToString();
23       hint3.text = checkkey.ToString();
24
25   }
```

Kode 3.5: Implementasi pertanyaan pada game

Implementasi kesulitan dilakukan dengan membuat target jatuh pada lokasi tertentu, cara penentuan dimana target akan di munculkan dilakukan dengan memberikan area secara eksplisit dalam kode, namun program akan mengambil point dari area yang sudah ditentukan secara acak, seperti yang terlihat pada Kode

3.6.

```
    void CreateTargetCube()
    {
        if (islevel1Started)
        {
            int randSpawnVocals = Random.Range(-990, -950);
            Vector3 spawnpos1 = new Vector3(randSpawnVocals, 10,
    5);

            intervalTime -= Time.deltaTime;
            if (intervalTime <= 0)
            {
                var target = Instantiate(targetCube, spawnpos1,
    this.transform.rotation);
                Rigidbody rb = target.GetComponent<Rigidbody>();
                intervalTime = 8f;
            }

        }
        if (islevel2Started)
        {
            int randSpawnVocals = Random.Range(-790, -750);
            Vector3 spawnpos1 = new Vector3(randSpawnVocals, 10,
    30);

            intervalTime -= Time.deltaTime;
            if (intervalTime <= 0)
            {
                var target = Instantiate(targetCube, spawnpos1,
    this.transform.rotation);
                Rigidbody rb = target.GetComponent<Rigidbody>();
                intervalTime = 8f;
            }
        }
        if (islevel3Started)
        {
            int randSpawnVocals = Random.Range(-590, -550);
            Vector3 spawnpos1 = new Vector3(randSpawnVocals, 10,
    60);

            intervalTime -= Time.deltaTime;
            if (intervalTime <= 0)
            {
```

```
38                  var target = Instantiate(targetCube, spawnpos1,
    this.transform.rotation);
39                  Rigidbody rb = target.GetComponent<Rigidbody>();
40                  intervalTime = 8f;
41              }
42          }
43
44
45      }
```

Kode 3.6: Implementasi tingkat kesulitan

Kode 3.6 merupakan template yang digunakan untuk membuat blok *answer* untuk huruf vokal, kode yang sama dengan pergeseran lokasi pembuatan digunakan untuk membuat blok *answer* bertipe tanda baca dan huruf non-vokal.

Untuk menghindari pembuatan blok *answer* tidak terlalu banyak sehingga menyebabkan performa *game* untuk menurun, maka setiap blok *answer* yang dibuat akan hancur setelah melewati 10 detik dihitung dari saat blok tersebut dibuat oleh program. Selain itu, blok juga akan hancur apabila ditembak oleh *player*, pada saat hancur nilai huruf dari blok yang hancur akan ditampilkan di layar. Implementasi hancur berdasarkan waktu dapat dilihat pada kode 3.7 dan implementasi hancur saat terkena peluru dari *player* dapat dilihat pada kode 3.8.

```
1       IEnumerator SelfDestruct()
2       {
3           yield return new WaitForSecondsRealtime(10);
4           Destroy(gameObject);
5       }
```

Kode 3.7: Implementasi Self-destruct pada blok yang dibuat

```
1       private void OnCollisionEnter(Collision collision)
2       {
3           if (collision.gameObject.CompareTag("Bullet"))
4           {
5               Debug.Log(cubeVal);
6               Destroy(gameObject);
7               testlogic.GetComponent<GameLogic>().ShowDestroyedCube(
    cubeVal);
8
9           }
10      }
```

Kode 3.8: Implementasi destruct on collission

Pada tiap *level*, apa bila *player* telah menembak blok *answer* sebanyak jumlah huruf dalam kata yang menjadi pertanyaan, maka akan dilakukan pengecekan jawaban, jika *player* berhasil menjawab pertanyaan dengan benar, log-huruf yang sudah dijawab akan direset dan *player* akan dapat kembali berjalan untuk pindah ke area *level* berikutnya. Jika *player* menjawab dengan salah, maka *player* akan diarahkan ke layar *Game Over*. Implementasi pengecekan jawaban dapat dilihat pada Kode 3.9

```
        if(characterControl.isLevel1Started)
        {
            timeSpent = Time.time;
            string correctAns = DecodeCaesar(pickedquestion1,
    checkkey);
            if (ans == correctAns)
            {
                isLevel1Cleared = true;
                characterControl.ismoveable = true;
                characterControl.ToggleActiveCamera();
                stage1ClearTime = CountClearTime(Time.time);
                stage1answer = correctAns;
                finalStage1CTime =  GetFinalTime(stage1ClearTime);
                cvlog = "";
                CubeLog.SetText(cvlog);
            }
            else
            {

                SceneManager.LoadScene("GameOver");
            }

        }
```

Kode 3.9: Implementasi cek jawbaan pada level 1

Pada level 2 dan level 3, dilakuakan pengecekan jawaban dengan konsep yang sama, namun pada level 3, apa bila *player* menjawab dengan benar, maka akan diarahkan ke layar menang. Pada saat *player* menyelesaikan level tiga akan dilakukan penyimpanan *clear data* ke *device*.

Clear Data yang disimpan dalan *device* adalah data kata-kata yang telah ditemukan dalam *stage*, jumlah kata yang telah ditemukan dalam *stage*, waktu menyelesaikan *stage*, dan juga tanggal menyelesaikan *stage*. Implementasi saat pertama kali melakukan penyimpanan *Clear Data* dapat dilihat pada Kode 3.10 dan

untuk implementasi penyimpanan untuk menimpa *Clear Data* yang sudah ada dapat dilihat pada Kode 3.11.

```
        if(loadedData == null )
        {
            playData.allwordFound = new string[3];
            playData.allwordFound = stageanswers;
            playData.allwordCount = stageanswers.Length;
            stageClearRecordItem.wordsFoundThisStage =
    stageanswers;
            stageClearRecordItem.stageClearTime =
    finalLevelClearTime;
            stageClearRecordItem.clearDate = DateTime.Now;
            ClearRecordItems.Add(stageClearRecordItem);
            playData.clearRecordItems = ClearRecordItems;
            SerializeJson(playData);
        }
```

Kode 3.10: Implementasi penyimpanan pertama

Pada implementasi penyimpanan yang menimpa data sebelumnya dilakukan proses duplikasi dari list data kata-kata yang sudah ditemukan untuk ditambahkan dengan kata-kata yang baru saja ditemukan pada sesi *stage* yang baru diselesaikan.

```
            ClearRecordItems newRecordItems = new ClearRecordItems
    ();
            string[] newAddedWords = new string[loadedData.
    allwordFound.Length + stageanswers.Length];
            loadedData.allwordFound.CopyTo(newAddedWords, 0);
            stageanswers.CopyTo(newAddedWords, loadedData.
    allwordFound.Length);
            string[] uniqueAddedWords = newAddedWords.Distinct().
    ToArray();
            playData.allwordFound = uniqueAddedWords;
            playData.allwordCount = uniqueAddedWords.Length;
            newRecordItems.wordsFoundThisStage = stageanswers;
            newRecordItems.stageClearTime = finalLevelClearTime;
            newRecordItems.clearDate = DateTime.Now;
            ClearRecordItems.Add(newRecordItems);
            playData.clearRecordItems = ClearRecordItems;
            SerializeJson(playData);
```

Kode 3.11: Implementasi penyimpanan diatas file yang sudah ada

Metode penyimpanan yang digunakan pada *game* ini adalah metode Serialisasi JSON. Seperti yang tertulis pada Kode 3.10 dan Kode 3.11, metode Serialisasi JSON dipanggil dengan memberikan input berupa data yang akan di simpan pada *device*.

3.4.3 Implementasi Achievement

Pada tahap terakhir ini, insentif untuk membuat pemain tertarik untuk menyelesaikan teka-teki hingga sempurna dirancang, implementasi dari insentif ini berbentuk ranking kecepatan menyelesaikan *stage* dan juga jumlah total kata yang ditemukan di tiap *stage*, jika berhasil menemukan semua kata-kata yang disediakan sebagai pertanyaan, *player* akan dapat membaca latar belakang karakter yang digunakan.

Implementasi pengukuran kecepatan menyelesaikan *stage* dapat dilihat pada Kode 3.12. Perhitungan waktu ini dilaksanakan baik saat *player* menyelesaikan *level* maupun *stage*, Input yang dimasukkan pada modul perhitungan waktu adalah waktu *player* menyelesaikan *level* maupun saat *player* menyelesaikan *stage*.

```
1     private int CountClearTime(float stageClearTime)
2     {
3         float ctime = stageClearTime - timeSpent;
4         return Mathf.FloorToInt( ctime);
5     }
6
7     public string GetFinalTime(int timetocount)
8     {
9         int mm = timetocount / 60;
10        int ss = timetocount;
11        int ms = timetocount * 1000;
12        string formattedTime = mm + ": " + ss + " : " + ms + ".";
13
14        return formattedTime;
15    }
```

Kode 3.12: Implementasi perhitungan waktu selesai

Ranking dari waktu selesai permainan akan ditampilkan pada layar pemilihan stage, Implementasi dapat dilihat pada Kode 3.13.

```
1     foreach ( var item in playData.clearRecordItems)
2     {
```

```
        formattedwords = string.Join(" - ", item.
    wordsFoundThisStage);
        GameObject recordItemSlot = Instantiate(itemTemplate,
    itemContainer);
        recordItemSlot.transform.GetChild(0).GetComponent<
    TextMeshProUGUI>().text = i.ToString();
        recordItemSlot.transform.GetChild(1).GetComponent<
    TextMeshProUGUI>().text = formattedwords;
        recordItemSlot.transform.GetChild(2).GetComponent<
    TextMeshProUGUI>().text = item.stageClearTime;
        recordItemSlot.transform.GetChild(3).GetComponent<
    TextMeshProUGUI>().text = item.clearDate.ToString();
        i = i + 1;
    }
```

Kode 3.13: Implementasi ranking

Selain kecepatan menyelesaian *stage*, *Achievement* juga dirancang dalam bentuk memberikan reward berupa cerita latar belakang karakter saat semua kata yang menjadi pertanyaan pada tiap *stage* sudah ditemukan, Implementasi dapat dilihat pada Kode 3.14.

```
    if (playData.allwordCount != stageOneUniqueWords.Length)
    {
        stageDetailButton.interactable = false;
    }
    else
    {
        stageDetailButton.interactable = true;
    }
```

Kode 3.14: Implementasi achievement berdasarkan jumlah Kata

3.5 Pengujian Aplikasi *Game*

Setelah aplikasi *game* selesai dibangun, dilakukan uji kelayakan aplikasi*game*, pengujian dilakukan menggunakan uji *black box*, yaitu pengujian aplikasi dimana penguji tidak terlibat dalam pembangunan aplikasi *game*. Pengujian ini berfungsi untuk memastikan aplikasi *game* dapat berjalan sesuai dengan apa yang diharapkan.

3.6 Pengumpulan dan Pengolahan Data Kepuasan Pemain

Setelah aplikasi *game* diuji, aplikasi *game* disebarkan kepada pemain, dan pemain akan diberikan *form feedback* untuk memberikan tanggapan, tanggapan dari pemain akan diolah untuk mencari apakah tujuan penelitian tercapai menggunakan sistem *USE Questionnaire* yang diberikan saat *player* menyelesaikan *game* baik secara sempurna maupun saat pertama kali menyelesaikan. Pada kuisioner ini *feedback* termasuk kemudahan memainkan *game* dan pemahaman *player* terhadap latar belakang dari karakter yang dimainkan.

Pertanyaan yang diberikan pada pemain terbagi menjadi lima kategori yaitu kategori general, untuk mengelompokkan jenis pemain yang memainkan *game*, kategori *Usefullness* untuk mengukur seberapa berfungsinya *game* untuk mencapai tujuan yang diharapkan, kategori *Ease of Use* untuk mengukur semudah apa *game* dimainkan oleh pemain, kategori *Ease of Learning* untuk mengukur semudah apa kontrol dari *game* dapat dipahami oleh pemain dan kategori *Satisfaction* untuk mengukur kepuasan pemain terhadap *game*,

Pertanyaan yang diberikan dapat dilihat pada tabel 3.1:

Tabel 3.1. Tabel pertanyaan dalam Kuisioner

Category	**Question**
Control Question	In what year you are born?
	How accustomed you are to playing game in general?
	Have you ever heard of Caesar Cipher before playing this game?
	In general, would you consider yourself a caring person?
Usefulness	The game helps me to understand the existence of message behind what is seen.
	The game makes me realize how important it is to think before saying.
	The game leads me into trying to understand a story better than just reading what was written.
	The game helps me understand more about Caesar Shift.
Ease of Use	How easy is it to navigate the menu?
	How easy is it to control the player character?
	How difficult is the question on each stage?
	How easy is it to play the game?
	How are the overall gameplay impression of the game?
Ease of Learning	How easy is it to understand the concept of the game?
	How easy is it to understand the tutorial provided in-game?
	How easy is it to learn the pattern in the question?
Satisfaction	I would play the game again to reach 100% completion
	I would recommend the game to friends
	I had fun playing the game
	I learn something new by playing the game
	The game was capable of providing fun while also helps to player to learn
	The game was capable of conveying what the creator want to tell to the player.

Untuk setiap pertanyaan yang tertera pada tabel 3.1, apa bila pertanyaan memiliki jawaban bersifat skalar, maka jawaban akan berbentuk *Likert Scale*, dimana untuk setiap poin akan terdapat skor *likert* yang akan digunakan untuk menghitung skor akhir baik per-orang untuk menentukan kepuasan akhir, ataupun

untuk menghitung rasio penjawab pada tiap pertanyaan. Setelah mendapatkan hasil skor *likert* dapat dihitung nilai untuk menentukan pada kategori kepuasan apa pemain tersebut berada.

Untuk mendapatkan nilai distribusi jawaban per-pertanyaan yang menggunakan *Likert Scale* dapat dilakukan dengan cara menghitungan nilai *Likert* yang didapat dengan mengalikan jumlah pemain yang memilih dengan skor opsi *Likert* tersebut, kemudian mentotal semua nilai hasil kali pemain dan skor *likert* pada pertanyaan tersebut, setelah itu memasukkan nilai total yang didapat ke rumus interpretasi pada persamaan 4.6. Nilai interpretasi setelah itu dicocokkan dengan tabel nilai *treshold* untuk mendapatkan impresi rata-rata pemain terhadap argumen dalam pertanyaan tersebut.

3.7 Penulisan Laporan

Hasil Perancangan dan Pembangunan Aplikasi *Game* akan didokumentasikan dalam bentuk laporan sebagai bukti pengerjaan penelitian.