Pengaruh Refactoring Terhadap Tingkat Pemeliharaan Perangkat Lunak Pada Aplikasi Permainan "Infection Defender"

Siti Halimatus Sadiyah[1], Anita Qoiriah[2]

[1,2] Teknik Informatika, Fakultas Teknik, Universitas Negeri Surabaya
[1]siti.19065@mhs.unesa.ac.id
[2]anitaqoiriah@unesa.ac.id

***Abstrak*— Dalam era kemajuan teknologi yang pesat, sistem perangkat lunak semakin kompleks, menghadirkan tantangan dalam meminimalisir efek sampingnya. Kualitas perangkat lunak menjadi fokus penting, dan salah satu atribut kualitas yang relevan adalah *maintainability* (kemampuan pemeliharaan). Penelitian ini mengkaji pengaruh teknik *refactoring* terhadap tingkat pemeliharaan perangkat lunak, dengan menggunakan aplikasi permainan *"infection defender"* sebagai objek penelitian. *Refactoring* dianggap kunci dalam meningkatkan struktur kode. tujuan dari penelitian ini yaitu untuk mengetahui pengaruh dari teknik *refactoring* terhadap tingkat pemeliharaan perangkat lunak.**

Metode penelitian yang digunakan adalah metode kuantitatif. Tahapan ataupun proses yang digunakan dalam penelitian ini sesuai model DSRM (*Design Science Research Methodology*) yang diajukan oleh Peffers. Sebelum dilakukannya *refactoring* yakni melakukan pengelompokkan *code smell* sesuai dengan buruknya kode yang terdeteksi. Selanjutnya di dapatkan hasil *refactoring* dengan menggunakan beberapa teknik yang berbeda dan dilakukan pengukuran nilai tingkat pemeliharaan perangkat lunak berdasarkan nilai metriknya yaitu *maintainability index, cyclomatic complexity, depth of inheritance, class coupling, lines of source code* dan *lines of executable code.*

Hasil implementasi dan pengujian dapat diperoleh kesimpulan bahwa penerapan teknik *refactoring* berdasarkan nilai metrik dapat meningkatkan kualitas perangkat lunak dan mengurangi serta menghindari adanya terjadi *bug* dengan selisih nilai + 3,5%.

Kata Kunci*— *Refactoring, Maintainability,* kualitas perangkat lunak, *code smell.

I. PENDAHULUAN

Seiring berjalannya waktu dengan adanya kemajuan teknologi yang pesat dan kebutuhan pengguna yang meningkat, tentunya kompleksitas sistem pada perangkat lunak akan terus mengalami peningkatan. Fenomena tersebut menjadi tantangan, terlebih semakin tingginya kompleksitas sebuah sistem maka akan semakin sulit sistem tersebut untuk meminimalisir efek sampingnya.[1]

Indikator perangkat lunak yang baik dapat dilihat dari kualitas perangkat lunak. Menurut International Organization for Standardization atau yang biasa disebut dengan ISO, kualitas perangkat lunak dapat dievaluasi dari sisi internal dan eksternal. Terdapat delapan atribut untuk mengukur kualitas perangkat lunak, salah satunya adalah atribut *maintainability*[2].

Maintainability (pemeliharaan perangkat lunak) merupakan tingkat efektivitas dan efisiensi yang dimana suatu sistem dapat dimodifikasi lebih lanjut. *Maintainability* juga bisa diartikan sebagai kemampuan suatu sistem yang dapat dipelihara dan diperbaiki[3]. Dengan melakukan perhitungan nilai *maintainability* pada perangkat lunak tentunya bisa membantu memutuskan apakah perangkat lunak yang di analisis mudah dirawat atau harus dilakukan perancangan ulang.

Hal yang mempengaruhi *maintainability* antara lain *refactoring, design pattern, code smell,* dan *aesthetic defects*. Salah satu faktor yang menarik adalah *refactoring*. Mengingat *refactoring* dilakukan untuk meningkatkan struktur kode internal menjadi lebih baik dan tetap mempertahankan fungsinya. Dan struktur kode yang buruk adalah pendorong utama di balik dilakukannya pemfaktoran ulang atau *refactoring*[4]. Secara umum, kegiatan *refactoring* individu memiliki efek pada sebagian besar atribut kualitas perangkat lunak salah satunya *maintainability* yang dieksplorasi dalam studi primer. Dan dalam hal ini teknik *refactoring* sangat berpengaruh pada pemeliharaan setiap sistem. Semakin tinggi skor *maintainability*nya maka semakin bagus juga aplikasi atau sistem tersebut[5].

Dalam hal ini, aplikasi permainan yang diambil untuk penelitian ini yakni *"infection defender"* yang dimana *game* ini sendiri sebelumnya telah dibahas pada penelitian sebelumnya dan untuk *source code*nya bersifat terbuka sehingga memudahkan untuk mengaksesnya dan melakukan *refactoring*[6]. Latar belakang memilih game ini untuk diangkat dalam penelitian tentunya didasari atas pernyataan pengembangan dan peneliti pada repositori sumber kodenya yang menyatakan bahwa kode tersebut masih berantakan serta kurangnya *refactoring.*

Berdasarkan pemaparan latar belakang di atas harapannya peneliti dapat memberikan kontribusi berupa pengetahuan baru terkait *refactoring* dan *maintainability*. Dengan adanya penelitian ini dapat membantu pemeliharaan terhadap aplikasi yang membutuhkan *maintenance* atau pemeliharaan lebih lanjut. Melakukan perawatan pada aplikasi, tentunya bisa mengurangi tingkat kerusakan serta menghindari kerugian biaya yang cukup besar. Selain itu, penelitian ini dapat dijadikan penelitian selanjutnya dan memberikan kontribusi serta gambaran terkait teknik *refactoring* untuk para pengembang perangkat lunak.

II. LANDASAN TEORI

A. *Maintainability*

Tingkat kemampuan pemeliharaan (*maintainability*) merupakan tingkat efektifitas dan efisiensi dimana produk atau sistem dapat dimodifikasi dan sejauh mana sistem tersebut bisa dipelihara serta diperbaiki. Kegiatan pemeliharaan perangkat lunak sendiri dimulai setelah rilis pertama dikirimkan kepada pengguna akhir (*user*).

B. *Code Smell*

Code smell adalah pola struktur kode program yang buruk sehingga terindikasi adanya suatu masalah dan membuat programmer sulit untuk memahami, membaca dan memodifikasi codingan ketika akan melakukan perubahan.

C. *Refactoring*

Refactoring didefinisikan sebagai suatu proses yang bertujuan untuk meningkatkan kualitas desain internal sistem perangkat lunak tanpa mengubah perilaku eksternalnya[7]. Dalam kata lain, struktur internal perangkat lunak dapat diperbaiki dengan *refactoring* tanpa membuat fungsionalitas baru. Strategi ini dapat dicapai dengan merestrukturisasi kelas, metode dan variabel terutama untuk membantu modifikasi dan ekstensi di masa mendatang.

D. Aplikasi Permainan *(Game)*

Aplikasi permainan *(game)* didefinisikan sebagai permainan yang menggunakan media elektronik. *Game* merupakan permainan yang diciptakan untuk pembelajaran yang dimana selain bisa menjadi sarana hiburan, akan tetapi diharapkan bisa mengedukasi dan menambah wawasan pengetahuan. *Game* diciptakan semenarik mungkin agar bisa menarik perhatian pengguna dan membuatnya mendapatkan kepuasan batin.

E. *C# (C Sharp)*

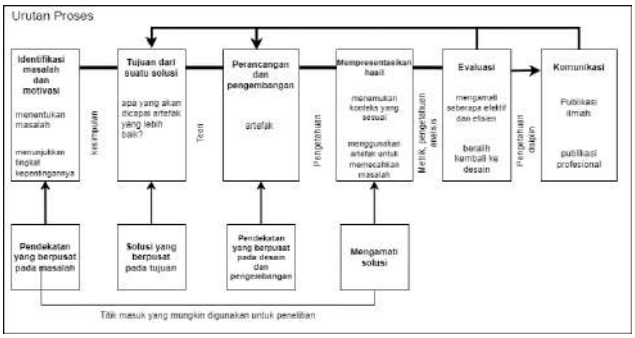
C# *(C Sharp)* merupakan sebuah bahasa pemrograman berorientasi objek, sebagai bagian dari inisiatif kerangka NET Framework yang dikembangkan oleh Microsoft. C# bahasa pemrograman kombinasi dari C++ dan *visual basic* yang digunakan untuk pengembangan *game,* yang dimana bahasa C# juga terintegrasi dengan software Unity yang digunakan untuk pembuatan model 2D dan 3D dan penerapan C# digunakan untuk pengembangan pembuatan aplikasi ini[8].

F. *Unity*

Unity merupakan mesin pengembang *game* yang digunakan untuk pembuatan video *game* dan mengekspornya untuk beberapa platform. Unity terdiri dari mesin grafis, mesin fisik dan pratinjau permainan langsung dimana perubahan yang dilakukan dilihat dalam fase pemrograman secara real-time. Ini merupakan satu set lengkap untuk membuat video *game* dan proyek interaktif lainnya, menyederhanakan proses pengembangan dan membuatnya lebih cepat.

III. METODOLOGI PENELITIAN

Metodologi penelitian yang akan digunakan pada penilitan ini adalah metode kuantitatif. Adapun desain penelitian yang digunakan yakni *Design Science Research Methodology (DSRM)* atau desain sains penelitian.



Gambar 1 Tahapan Proses DSR

Gambar 1 diatas merupakan alur tahapan proses yang dilakukan dalam penelitian ini. Dengan serta tahapannya akan dijelaskan seperti berikut ini.

A. Identifikasi masalah dan motivasi

Pada tahapan pertama diawali dengan mengidentifikasi masalah yang ada pada topik yang diangkat dan membenarkan nilai solusi. Dalam hal ini penentuan sebuah masalah yang digunakan untuk pengembangan tentunya harus ada solusi artifaktual yang efektif untuk mengurangi masalah secara konseptual. Penggunaan solusi juga dapat menangkap kompleksitas masalah yang ada. Hal ini menjadi poin penting adanya solusi dalam pengidentifikasian sebuah masalah.

B. Tujuan dari suatu Solusi

Pada tahapan selajutnya yakni menyimpulkan tujuan solusi dari tahapan sebelumnya yakni pengidentifikasian masalah. Tujuannya bisa bersifat kuantitatif atau kualitatif. Apabila bersifat kuantitatif contohnya yakni istilah dimana solusi yang diinginkan akan lebih baik dari pada yang sekarang, dan jika bersifat kualitatif contohnya yakni dimana artefak baru diharapkan untuk mendukung solusi untuk masalah yang sampai sekarang tidak ditangani. Tahapan tujuan ini harus disimpulkan secara rasional dari spesifikasi masalah seperti sumber daya yang diperlukan termasuk pengetahuan tentang keadaan masalah dan solusi saat ini.

C. Perancangan dan pengembangan

Pada tahapan ini yakni proses awal mulai perancangan dan pengembangan atau tahap modifikasi. Kegiatan yang dilakukan dalam tahapan ini yakni menentukan fungsionalitas artefak yang diinginkan dan arsitekturnya, lalu membuat artefak yang sebenarnya. Dalam tahap

pengembangan yang dilakukan untuk penelitian ini yakni menggunakan teknik *refactoring.*

D. Mempresentasikan hasil

Tahapan selanjutnya yakni mempresentasikan hasil, yang dimana ketika proses perancangan dan pengembangan sebelumnya sudah selesai maka melakukan pengetesan untuk melihat perubahan yang terjadi dari sebelum adanya pengembangan dan sesudah dilakukannya pengembangan. Di tahapan ini tunjukkan tingkat efektivas artefak untuk memecahkan masalah yang diangkat. Hal ini nantinya akan berkaitan dalam hal eksperimen, simulasi, studi kasus, bukti atau aktivitas lain yang sesuai. Sumber daya yang diperlukan di tahapan ini mencakup pengetahuan yang efektif tentang cara menggunakan artefak untuk memecahkan masalah ini.

E. Evaluasi

Selanjutnya tahapan evalusi, di tahapan ini perlu dilakukan pengamatan dan pengukuran seberapa baik atau berpengaruh artefak mendukung solusi untuk masalah ini. Kegiatan ini membandingkan tujuan solusi dengan hasil pengamatan aktual dari penggunaan artefak di tahapan sebelumnya. Dalam hal ini membutuhkan pengetahuan tentang metrik yang relevan dan teknik analisis. Tahapan evaluasi dapat mencakup item seperti perbandingan fungsionalitas artefak, tujuan ukuran kinerja kuantitatif seperti anggaran atau item yang menghasilkan kepuasan survei. Di tahapan ini peneliti dapat memutuskan apakah akan mengulangi tahapan ketiga di tahapan proses pengembangan untuk mencoba meningkatkan keefektifan artefak atau melanjutkan ke tahapan selanjutnya yakni tahapan komunikasi.

F. Komunikasi

Selanjutnya tahapan komunikasi, di tahapan ini komunikasikan masalah dan kepentingannya, kegunaan dan keterbaruannya, kekakuan desainnya, dan keefektifannya kepada peneliti lainnya dan audiens lainnya yang relevan seperti dosen.

Gambaran Game

Berikut gambaran singkat dan tangkapan layar mengenai *game "infection defender",* antara lain :

1. Tampilan Main Menu

Pada tampilan gambar 2, terdapat beberapa fitur yang ada dalam aplikasi yakni *start, load, how to, settings, credits, quit* dan *highscore.*

Gambar 2 Tampilan Main Menu

2. Tampilan *Action*

Pada tampilan action gambar 3, pemain diminta untuk memperoleh poin sebanyak banyaknya dengan menggerakan papan kayu yang berada ditengah, tujuannya untuk menampung karakter manusia yang berjatuhan. Kemudian pemain harus menangkap karakter yang berjatuhan dari klinik dan membawanya ke rumah sakit, apabila karakter tidak tertangkap karena kemampuan kecepatan dalam menggerakkan dan mengantar ke rumah sakit kurang, maka nantinya karakter tersebut bisa disimpulkan meninggal dan menambah skor "infected people". Jika karakter memiliki gejala ringan dan tidak cukup parah, maka nantinya hanya perlu perawatan di klinik terdekat. Game ini akan berakhir sesuai dengan waktu (hari) yang dipilih untuk dimainkan.

Gambar 3 Tampilan Action

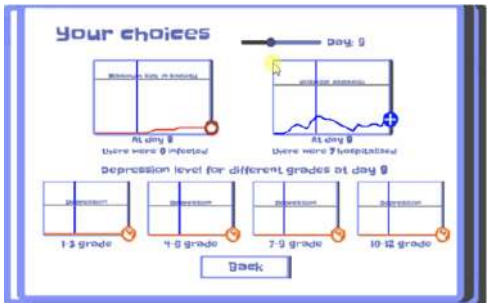
3. Tampilan Statistik

Pada tampilan ini merupakan hasil rekapitulasi setelah permainan berakhir. Contohnya seperti gambar 4 yang dimana ada 28 orang masuk rumah sakit dan 4 orang tidak tertolong sehingga menyebabkan meninggal dunia.

Gambar 4 Tampilan Statistik

4. Tampilan Pemilihan Level

Pada tampilan gambar 5, pemain bisa mengatur waktu (hari) yang ingin dimainkan dengan minimal waktu 1 hari dan maksimal 25 hari.

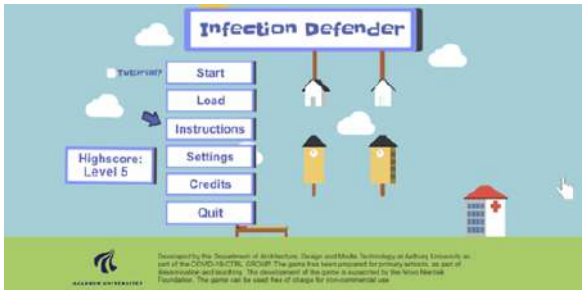


Gambar 5 Tampilan Pemilihan Level

IV. HASIL DAN PEMBAHASAN

A. Implementasi Aplikasi

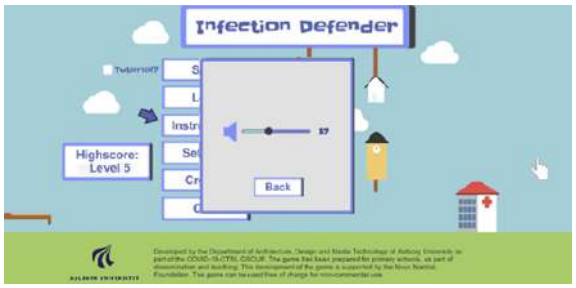
Dalam hal ini menjelaskan mengenai *game* secara terperinci serta menampilkan semua menu yang ada pada gambar 6, 7, 8, 9, 10 dan 11. Pada penelitian dan penerapan *refactoring* yang dilakukan menjelaskan bahwa hal ini tidak mempengaruhi dari segi tampilan atau sisi ekternal. Berikut merupakan tampilan *game* setelah dilakukan *refactoring.*



Gambar 7 Tampilan Main Menu

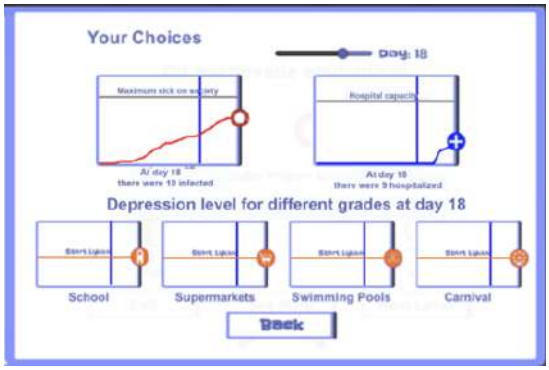
Gambar 6 Tampilan Action

Gambar 8 Tampilan Menu Instructions



Gambar 9 Tampilan Menu Settings

Gambar 10 Tampilan Statistik



Gambar 11 Tampilan Pemilihan Level

B. Pendeteksian *Code Smell*

Langkah awal sebelum dilakukannya refactoring yakni melakukan pengelompokkan code smell sesuai dengan buruknya kode. Pengidentifikasian code smell sendiri dilakukan dengan pemeriksaan secara manual. Berikut merupakan pengidentifikasian code smell pada aplikasi permainan ini, antara lain :

1. *Capture and Send Info*
   a. *Bloaters (Large Class)*

      Dalam class '*captureAndSendInfo'* pada kolom dibawah, memiliki banyak method dan variable yang dimana didalamnya memuat class yang cukup besar dan sulit dipahami serta dikelola.

```
public class captureAndSendInfo : MonoBehaviour
{
   // Start is called before the first frame update
   public static captureAndSendInfo current;
…
void showLastSave()
```

```
 {
    string showLastLog = "";
    showLastLog += " | " +
logs["GameVersion"][logs["GameVersion"].Count-
1].ToString();
      showLastLog += " | " +
logs["PlayID"][logs["PlayID"].Count-1].ToString();
 …
```

```
void eachDayData()
   {
      string currEventType = "dayPassed";
      string currGameState = "levelPlaying";
      gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
   }
```

```
void showStatisticsData()
   {
      string currEventType = "statisticsScreen";
      string currGameState = "showStatistics";
      gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
   }
```

```
void nextLevelData()
   {
      string currEventType = "nextLevel";
      string currGameState = "levelLoadNext";
      gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
      sendToServer();
   }
```

```
void levelEndData(int whatEnding)
   {
      string currEventType = "";
      if (whatEnding == 0)
      {
         currEventType = "levelWin";
      }
      else if (whatEnding == 1)
      {
         currEventType = "levelLoseHospital";
      }
      else if (whatEnding == 2)
      {
         currEventType = "levelLoseSociety";
      }
      else if (whatEnding == 3)
      {
         currEventType = "levelLoseSchool";
      }
    string currGameState = "levelEnd"
    gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
   }
```

b. *Dispansables (Duplicate Code)*

Banyak bagian code yang sama yang diulang dalam beberapa method seperti pada bagian *'restartLevelData'* , *'eachDayData'* dan '*showStatisticsData'* seperti pada kolom dibawah. Hal ini dapat dihindari dengan memisahkan code yang sama ke dalam method yang terpisah.

```
   void restartLevelData()
   {
      string currEventType = "restartLevel";
      string currGameState = "levelReload";
      gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
      sendToServer();
   }
```

```
void eachDayData()
   {
      string currEventType = "dayPassed";
      string currGameState = "levelPlaying";
      gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
   }
```

```
void showStatisticsData()
   {
      string currEventType = "statisticsScreen";
      string currGameState = "showStatistics";
      gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
   }
```

2. *Game End Initializer*

a. *Bloaters (Long Method)*

Dalam method '*Update ()*' yang terdapat pada kolom dibawah, memuat kode yang terlalu panjang atau memiliki kompleksitas kondisional yang tinggi (conditional complexity). Hal ini tentunya dapat mengurangi pemahaman alur logika yang ada dalam method '*Update ()*' tersebut. Kondisi kode yang seperti ini dapat lebih rentan terhadap adanya bug dan kesalahan logika.

```
void Update()
   {
      if (globalTimer.current.daysPassed >=
globalTimer.current.maxDays)
      {
         whatEnd = 0;
      }
      if
(globalScoreKeeper.current.numberSickHospital >
globalScoreKeeper.current.maxHospitalCapacity)
      {
         whatEnd = 1;
      }
      if
(globalScoreKeeper.current.numberSickSociety >
globalScoreKeeper.current.maxSickSociety)
      {
         whatEnd = 2;
      }
…
```

C. Implementasi *Refactoring*

Implementasi refactoring dilakukan setelah terindikasi adanya code smell. Penerapan refactoring yang digunakan yakni teknik composing method. Berikut merupakan hasil refactoring yang digunakan pada aplikasi permainan ini, antara lain :

1. *Capture and Send Info*
   a. *Simplifying conditional expressions (consolidate duplicate fragment)*

Berikut merupakan code sebelum dilakukannya refactoring seperti pada kolom dibawah ini.

```
void eachDayData()
   {
      string currEventType = "dayPassed";
      string currGameState = "levelPlaying";
      gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
   }
```

```
void showStatisticsData()
   {
      string currEventType = "statisticsScreen";
      string currGameState = "showStatistics";
      gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
   }
```

```
void nextLevelData()
   {
      string currEventType = "nextLevel";
      string currGameState = "levelLoadNext";
      gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
      sendToServer();
   }
```

```
void restartLevelData()
   {
      string currEventType = "restartLevel";
      string currGameState = "levelReload";
      gatherAndSaveData(currEventType,
currGameState);
      showLastSave();
      sendToServer();
   }
```

Dan berikut merupakan code setelah dilakukannya refactoring seperti pada kolom dibawah ini.

```
void eachDayData()
   {
      gatherAndSaveEventData("dayPassed",
"levelPlaying");
   }
```

```
void showStatisticsData()
   {
      gatherAndSaveEventData("statisticsScreen",
"showStatistics");
   }
```

```
void nextLevelData()
   {
      gatherAndSaveEventData("nextLevel",
"levelLoadNext");
      sendToServer();
   }
```

```
void restartLevelData()
   {
      gatherAndSaveEventData("restartLevel",
"levelReload");
      sendToServer();
   }
```

Penerapan refactoring yang dilakukan pada kolom diatas menjelaskan terkait penggabungan variabel (‘*currEventType’* dan *‘currGameState’*) menjadi argument langsung dalam pemanggilan method ‘gatherAndSaveEventData’ . Dengan melakukan penerapan ini, menghindari adanya duplikasi antara variabel-variabel tersebut.

b. Composing method (inline temp)

Penerapan refactoring yang dilakukan pada kolom diatas menjelaskan terkait variabel sementara (‘currEventType’ dan ‘currGameState’), kemudian diganti dengan expression yang sesuai untuk mengurangi adanya duplikasi code dan membuat code lebih bersih serta mudah dikelola.

2. *End Results Show*
   a. *Simplifying method calls (replace parameter with explicit methods)*

Berikut merupakan code sebelum dilakukannya refactoring seperti pada kolom dibawah ini.

```
if (checkEnding == 0)
    {
      buttonNext.SetActive(true);
      buttonRestart.SetActive(false);
    }
    else
    {
          buttonNext.SetActive(false);
          buttonRestart.SetActive(true);
    }
```

Dan berikut merupakan code setelah dilakukannya refactoring seperti pada kolom dibawah ini.

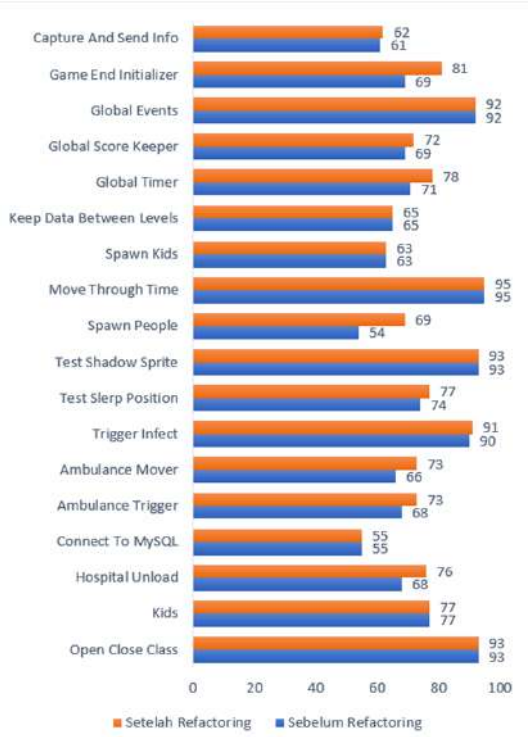
```
void updateButtonVisibility(int endingType)
   {
      buttonNext.SetActive(endingType == 0);
      buttonRestart.SetActive(endingType != 0);
   }
```

Dari hasil penerapan refactoring yang telah dilakukan pada kolom diatas menjelaskan bahwa parameter ‘*endingType’* secara eksplisit (jelas dan rinci) menentukan perilaku method ‘updateButtonVisibility’. Hal ini membuat method tersebut lebih bersifat umum dan memungkinkan penggunaannya dengan parameter yang berbeda tanpa perlu menggunakan kondisi percabangan *(if-else).*

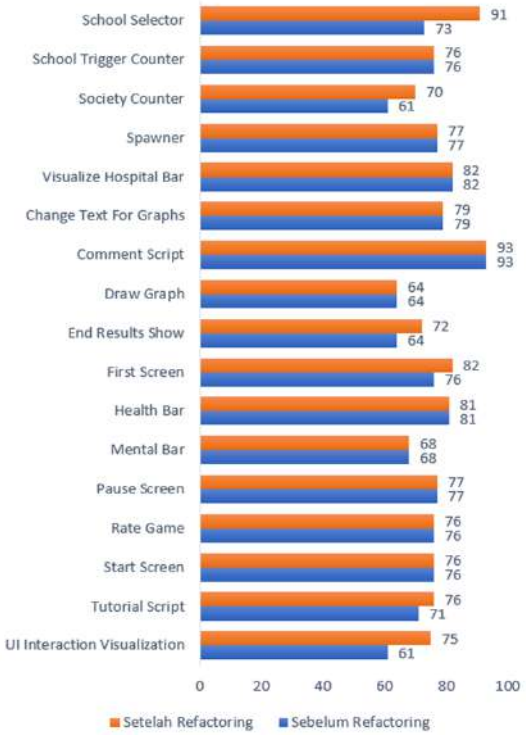
D. Pengukuran Nilai Tingkat Pemeliharaan Perangkat Lunak

a. Pengukuran nilai metrik *maintainability index*

Dalam maintainability index, ketika nilai indeks yang ditunjukkan berwarna hijau yakni dengan nilai antara 20 – 100, maka kode tersebut memiliki pemeliharaan kode yang baik . Untuk nilai indeks yang ditunjukkan warna kuning yakni dengan nilai antara 10 – 19, maka hal ini menunjukkan bahwa kode tersebut cukup dapat dipertahankan. Dan untuk nilai indeks yang ditunjukkan dengan warna merah yakni antara nilai 0 – 9, maka hal ini menunjukkan bahwa kode tersebut memiliki pemeliharaan yang rendah atau bisa dikatakan sebagai kode yang buruk. Berikut hasil dari penghitungan metrik ini menggunakan visual studio yang terdapat pada gambar 12 dan 13.



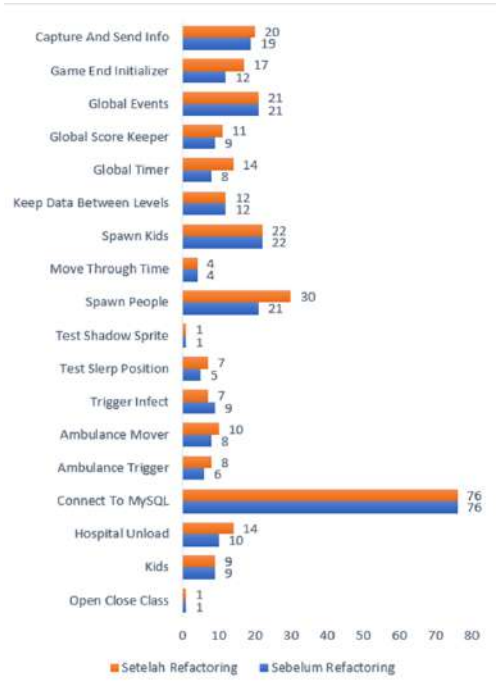
Gambar 12 Nilai Metrik Maintainability Index



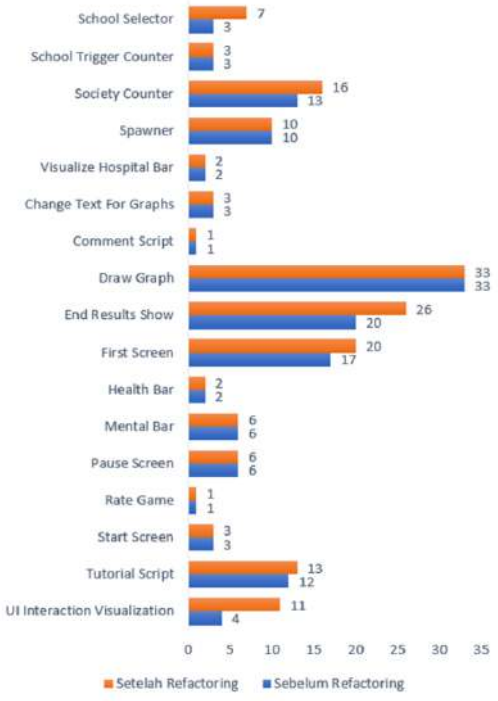
Gambar 13 Nilai Metrik Maintainability Index

b. Pengukuran nilai metrik *cyclomatic complexity*

Cyclomatic complexity merupakan metrik yang digunakan untuk mengukur kompleksitas struktural dari suatu program. Dalam penilaian pada metrik *cyclomatic complexity,* apabila nilai kompleksitas yang dihasilkan semakin tinggi maka akan semakin sulit untuk diuji dan dirawat. Jika nilai yang dihasilkan antara 1 – 10 maka memiliki tingkat resiko yang rendah dengan tipe prosedur sederhana, terstruktur dan stabil, lalu untuk nilai 11 – 20 maka memiliki tingkat resiko yang menengah dengan tipe prosedur yang lebih kompleks, lalu untuk nilai 21 – 50 memiliki tingkat resiko yang tinggi dengan tipe prosedur yang kompleks, sedangkan untuk nilai > 50 maka memiliki tingkat resiko yang sangat tinggi dengan tipe prosedur yang tentunya rentan akan kesalahan dan sangat mengganggu jika tidak diperbaiki. Berikut hasil dari penghitungan metrik ini menggunakan visual studio yang terdapat pada gambar 14 dan 15.



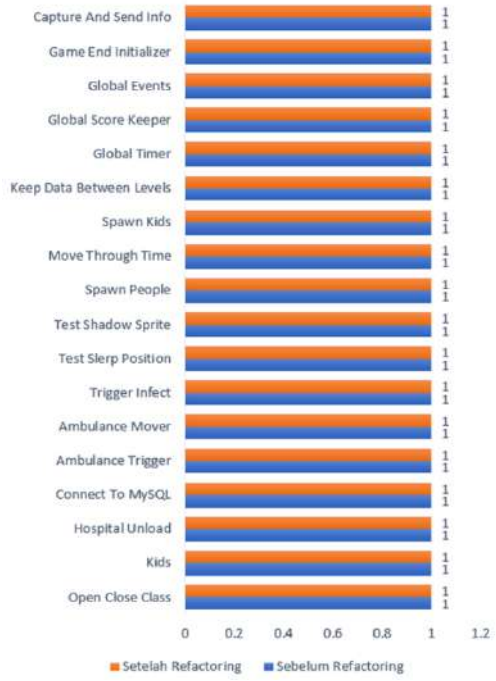
Gambar 14 Nilai Metrik Cyclomatic Complexity



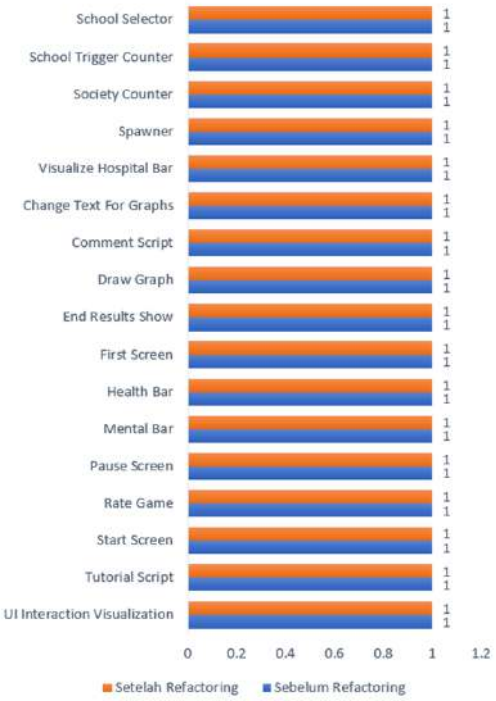
Gambar 15 Nilai Metrik Cyclomatic Complexity

c. Pengukuran nilai metrik *depth of inheritance*

Dalam penilaian metrik *depth of inheritance,* jika nilai yang dihasilkan semakin tinggi maka semakin tinggi juga potensi untuk modifikasi kelas dasarnya untuk menghasilkan perubahan yang signifikan, sedangkan jika nilai yang dihasilkan semakin kecil atau rendah maka dinilai baik. Pada gambar 16 dan 17 menunjukkan hasil dari penghitungan untuk metrik ini, dimana semua *class* menghasilkan nilai yang sama yakni 1 dan tidak mengalami perubahan apapun, dikarenakan *refactoring* yang dilakukan hanya berdasarkan *method* yang ada pada masing-masing *class* dan untuk menghindari adanya *refactoring* skala besar.



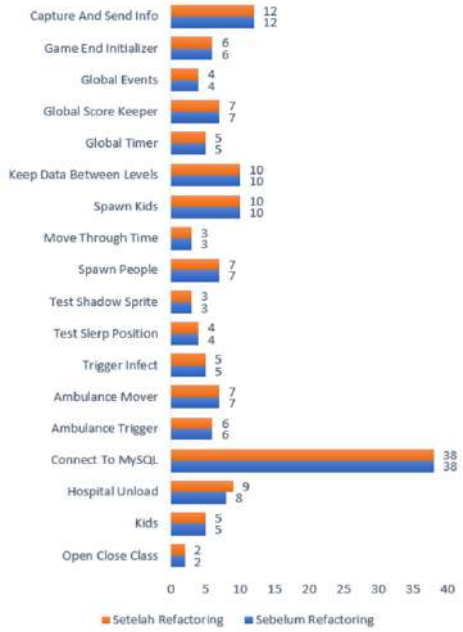
Gambar 16 Nilai Metrik Depth of Inheritance



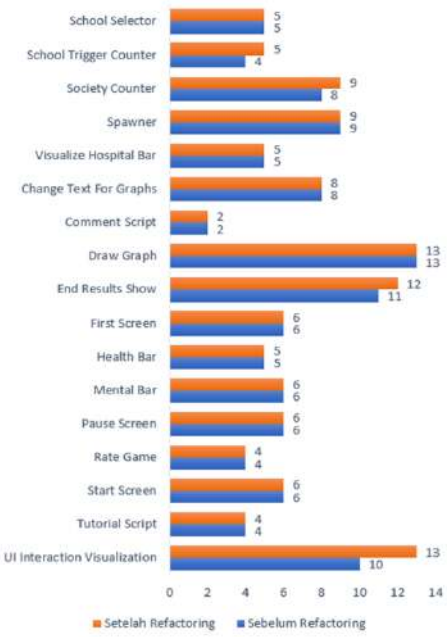
Gambar 17 Nilai Metrik Depth of Inheritance

d. Pengukuran nilai metrik *class coupling*

Dalam penilaian metrik *class coupling* ini penghitungan yang dilakukan berdasarkan tingkat ketergantungan atau hubungan dengan *class-class* yang lain. Penghitungan pada metrik ini tidak menggunakan rumus tertentu. Semakin tinggi angka yang dihasilkan maka semakin sulit ketika akan melakukan modifikasi terhadap class tersebut dikarenakan banyaknya ketergantungan antar class. Berikut hasil dari penghitungan metrik ini menggunakan visual studio yang terdapat pada gambar 18 dan 19.



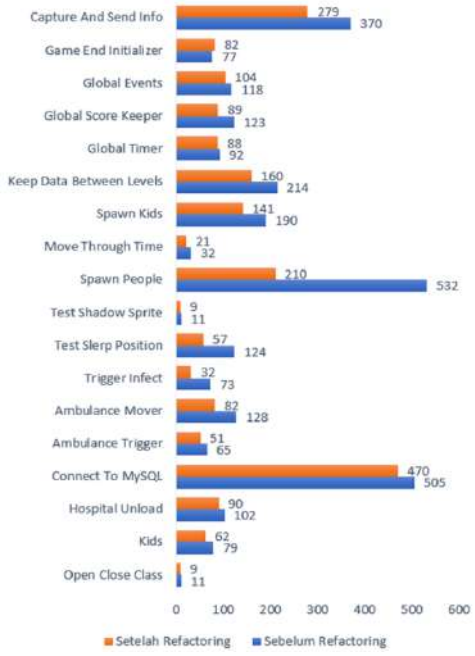
Gambar 18 Nilai Metrik Class Coupling



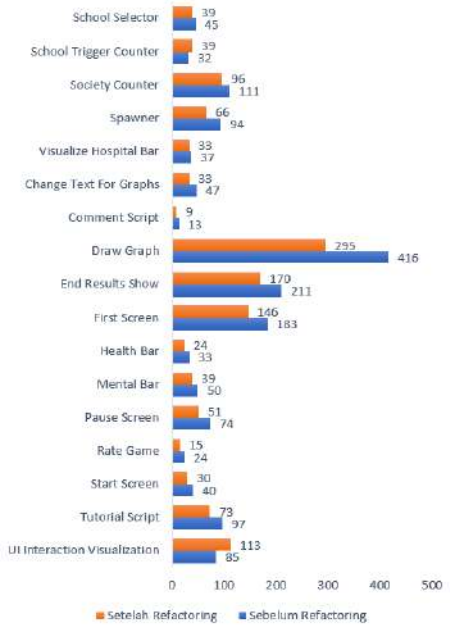
Gambar 19 Nilai Metrik Class Coupling

e. Pengukuran nilai metrik *lines of source code*

Dalam penilaian metrik *lines of source code* ini sebenarnya tidak diperlukan penghitungan menggunakan rumus tertentu, dikarenakan penghitungan ini dapat dilakukan secara manual hanya dengan menghitung jumlah baris code yang ada pada masing-masing class termasuk baris yang kosongpun ikut dihitungkan dalam perhitungannya. Berikut hasil dari penghitungan metrik ini menggunakan visual studio yang terdapat pada gambar 20 dan 21.



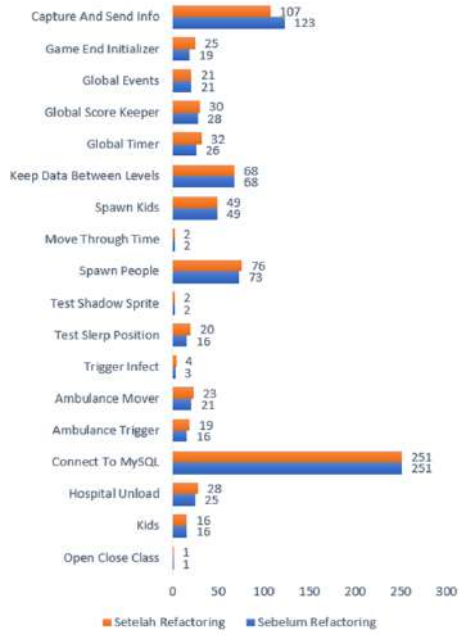
Gambar 20 Nilai Metrik Lines of Source Code



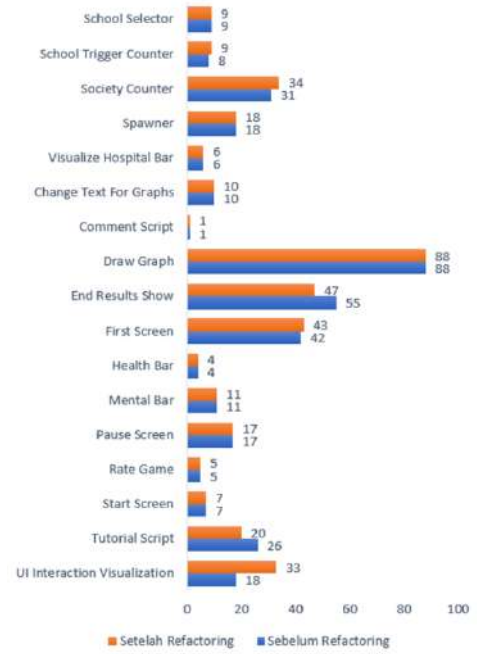
Gambar 21 Nilai Metrik Lines of Source Code

f. Pengukuran nilai metrik *lines of executable code*

Penghitungan metrik *lines of executable code* ini berbeda dengan penghitungan metrik *lines of source code*. Pada metrik ini penghitungannya yakni mengukur jumlah baris dalam *source code* yang berisi pernyataan atau instruksi yang dijalankan oleh mesin atau sistem dan untuk baris kosong, komentar atau baris yang hanya berisi spasi tidak termasuk dalam penghitungan metrik ini. Berikut hasil dari penghitungan metrik ini menggunakan visual studio yang terdapat pada gambar 22 dan 23.



Gambar 22 Nilai Metrik Lines of Executable Code



Gambar 23 Nilai Metrik Lines of Executable Code

Berikut contoh penghitungan manual yang dilakukan pada *class Game End Initializer* dan pada kolom dibawah ini merupakan *script* untuk *class* ini.

```
/// <summary>
/// The script keeps track if any of the level end conditions are reached. If they are the variable whatEnd is changed and the event levelEndingEvent are called.
/// Possible game endings:
/// - the last day of the level is reached - level is won - whatEnd == 0
/// - the number of sick in the hospital exceeds maximum capacity - level is lost - whatEnd == 1
/// - the number of sick in society exceeds the maximum possible - level is lost - whatEnd == 2
/// - the depression in any closed school exceeds the maximum possible - level is lost - whatEnd == 3
/// </summary>

public class gameEndInitializer : MonoBehaviour
{

    public static gameEndInitializer current;

    public int whatEnd = -2;

    GameObject[] allSpawners;

    int prevWhatEnd;

    private void Awake()
    {
        current = this;

        prevWhatEnd = whatEnd;
    }
```

```
    // Start is called before the first frame update
    void Start()
    {
        allSpawners =
GameObject.FindGameObjectsWithTag("spawners");
    }

    // Update is called once per frame
    void Update()
    {

        if (globalTimer.current.daysPassed >=
globalTimer.current.maxDays)
        {
            whatEnd = 0;
        }

        if (globalScoreKeeper.current.numberSickHospital >
globalScoreKeeper.current.maxHospitalCapacity)
        {
            whatEnd = 1;
        }

        if (globalScoreKeeper.current.numberSickSociety >
globalScoreKeeper.current.maxSickSociety)
        {
            whatEnd = 2;
        }

        bool isSchoolOverflow = false;
        for (int i = 0; i < allSpawners.Length; i++)
        {
            if
(allSpawners[i].GetComponent<spawner>().daysClosed >
globalScoreKeeper.current.maxDaysSchoolClosed)
            {
                isSchoolOverflow = true;
            }
        }

        if (isSchoolOverflow)
        {
            whatEnd = 3;
        }

        //Debug.Log(whatEnd + " | " + prevWhatEnd);
        if (whatEnd != prevWhatEnd && whatEnd>=-1 &&
prevWhatEnd >= -1)
        {
            GlobalEvents.current.levelEndingEvent(whatEnd);
        }


        prevWhatEnd = whatEnd;
    }
}
```

1. *Cyclomatic complexity*
   a. Penghitungan *cyclomatic complexity type awake ( )* pada tabel 1.

Table 1 Penghitungan type awake

Operation	Count
Procedure	1
CC (Total)	**1**

b. Penghitungan *cyclomatic complexity type start ( )* pada tabel 2.

Table 2 Penghitungan type start

Operation	Count
Procedure	1
CC (Total)	**1**

c. Penghitungan *cyclomatic complexity type update ( )* pada tabel 3.

Table 3 Penghitungan type update

Operation	Count
Procedure	1
if	6
for loop	1
Operator logika (&&)	2
CC (Total)	**10**

Ketentuan perhitungan *cyclomatic complexity* yakni menjumlahkan total per *type,* jadi total nilainya 12.

2. *Depth of inheritance*
   Penghitungan nilai *metrik depth of inheritance* hanya memiliki 1 nilai dikarenakan penghitungan ini dilakukan pada satu class yakni *class Game End Initializer.*

3. *Class coupling*
   Penghitungan nilai metrik *class coupling* dihitung dari berapa banyak *class* yang secara langsung bergantung dalam class ini. Hal ini mencakup variable instan, metode panggilan dan ketergantungan lainnya. Dalam hal ini, nama class yang berkaitan dengan class ini antara lain *class globalScoreKeeper, GameObject, globalTimer, allSpawners[i], spawner* dan *GlobalEvents*. Sehingga, jumlah nilai metrik class coupling ini yakni 6.

4. *Lines of source code*
   Penghitungan nilai metrik *lines of source code* dihitung berdasarkan jumlah baris *code* yang ada pada *class game end initializer* seperti pada kolom diatas**,** termasuk baris kosong dan komentarpun ikut dihitungkan dalam metrik ini. Pada *class* ini, terdapat 77 *LOC.*

5. *Lines of executable code*
   Penghitungan nilai metrik *lines of executable code* dihitung berdasarkan jumlah baris code yang berisi instruksi yang dapat dieksekusi oleh compiler. Nilai metrik ini diambil dari total keseluruhan per type yakni 18.

a. Penghitungan metrik *lines of executable code type awake ( )*

current = this;
prevWhatEnd = whatEnd;
Total = 2

b. Penghitungan metrik *lines of executable code type start ( )*

allSpawners = GameObject.FindGameObjectsWithTag("spawners");
Total = 1

c. Penghitungan metrik *lines of executable code type update ( )*

if (globalTimer.current.daysPassed >= globalTimer.current.maxDays)
whatEnd = 0;
if (globalScoreKeeper.current.numberSickHospital > globalScoreKeeper.current.maxHospitalCapacity)
whatEnd = 1;
if (globalScoreKeeper.current.numberSickSociety > globalScoreKeeper.current.maxSickSociety)
whatEnd = 2;
bool isSchoolOverflow = false;
for (int i = 0; i < allSpawners.Length; i++)
if (allSpawners[i].GetComponent<spawner>().daysClosed > globalScoreKeeper.current.maxDaysSchoolClosed)
isSchoolOverflow = true;
if (isSchoolOverflow)
whatEnd = 3;
if (whatEnd != prevWhatEnd && whatEnd>=-1 && prevWhatEnd >= -1)
GlobalEvents.current.levelEndingEvent(whatEnd);
prevWhatEnd = whatEnd;
Total = 15

6. *Maintainability Index*

a. Penghitungan *maintainability index type awake ( )* pada tabel 4.

Table 4 Penghitungan type awake

Type	Operators (n1)	Occurences (N1)	Operands (n2)	Occurences (N2)
Awake( ) : void	=	2	whatEnd	1
	;	2	current	1
			this	1
			prevWhatEnd	1
Jumlah	**2**	**4**	**4**	**4**

n = n1 + n2
= 2 + 4 = 6

N = N1 + N2
= 4 + 4 = 8

HV = N * log2 (n)
= 8 * log2 (6)
= 21

MI = MAX(0;171 – 5.2 * ln(HV) – 0,23 * (CC) – 16,2 * ln(LOC)) *100 / 171

= MAX(0;171 – 5.2 * ln(21) – 0.23 * (1) – 16.2 * ln(2)) *100/171
MI = 84

Keterangan :
HV = *Halstead Metrics Volumes*
CC = *Cyclomatic Complexity*
LOC = *Lines of Code*
perCM = *Percent line of comment*

b. Penghitungan *maintainability index type start ( )* pada tabel 5.

Table 5 Penghitungan type start

Type	Operators (n1)	Occurances (N1)	Operands (n2)	Occurances (N2)
Start() : void	=	1	GameObject.FindGameObjectsWithTag("spawners")	1
	;	1		
Jumlah	**2**	**2**	**1**	**1**

n = n1 + n2
= 2 + 1 = 3

N = N1 + N2
= 2 + 1 = 3

HV = N * log2 (n)
= 3 * log2 (3) = 5

MI = MAX(0;171 – 5.2 * ln(HV) – 0,23 * (CC) – 16,2 * ln(LOC)) *100 / 171

= MAX(0;171 – 5.2 * ln(5) – 0.23 * (1) – 16.2 * ln(1)) *100/171

MI = 95

c. Penghitungan *maintainability index type update ( )* pada tabel 6.

Table 6 Penghitungan type update

Type	Operators (n1)	Occurances (N1)	Operands (n2)	Occurances (N2)
Update () : void	{}	7	whatEnd	5
	int	1	current	8
	=	8	prevWhatEnd	3
	;	8	allSpawners.Length	1
	()		globalTimer.current.daysPassed	1
	if	6	globalTimer.current.maxDays	1
	>=	3	globalScoreKeeper.current.numberSickHospital	1
	>	3	0	2
	bool	1	globalScoreKeeper.current.maxHospitalCapacity	1
	for	1	globalScoreKeeper.current.maxSickSociety	1
	<	1	globalScoreKeeper.current.numberSickSociety	1
	++	1	1	1
	Length	1	allSpawners[i].GetComponent<spawner>().daysClosed	1
	[]	2	globalScoreKeeper.current.maxDaysSchoolClosed	1
	<>	1	2	1
	!=	1	isSchoolOverflow	3
	&&	2	FALSE	1
	levelEndingEvent	1	i	3
	GetComponent	1	GlobalEvents.current.levelEndingEvent(whatEnd)	1
			TRUE	1
			3	1
			-1	2
			GlobalEvents	1
Jumlah	**19**	**56**	**23**	**42**

n = n1 + n2
= 19 + 23 = 42

N = N1 + N2
= 56 + 41 = 97

HV = N * log2 (n)
= 97 * log2 (42) = 523

MI = MAX(0;171 – 5.2 * ln(HV) – 0,23 * (CC) – 16,2 * ln(LOC)) *100 / 171

= MAX(0;171 – 5.2 * ln(523) – 0.23 * (10) – 16.2 * ln(15)) *100/171

MI = 54

Penghitungan nilai metrik *maintainability index* dihitung rata-rata dari ketiga *type,* namun ketentuan penghitungan ini berlaku jika LOC nilainya lebih dari 2. Oleh karena itu, nilai yang diambil yakni dari *type awake* dan *update* dengan hasil nilai **69.**

V. KESIMPULAN

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan dapat diperoleh kesimpulan bahwa penerapaan

teknik *refactoring* berdasarkan nilai metrik seperti *maintainability index, cyclomatic complexity, depth of inheritance, class coupling, lines of source code* dan *lines of executable code* dapat meningkatkan kualitas perangkat lunak dan mengurangi serta menghindari adanya terjadi *bug.* Akan tetapi, tidak semua penerapan *refactoring* dapat dilakukan pada kesuluruhan kode. Adapun pemanfaatan penerapan teknik ini secara umum memberikan petunjuk kepada para pengembang untuk menulis kode suumber yang mudah dibaca agar pengembang lain dapat memahami maksud dari kode yang ditulis tersebut. Dimana Hasil *Refactoring* pada aplikasi *Infection Defender* pada metrik *Maintainability Index* mendapatkan nilai rata rata *Pra-Refactoring* 73.65 dan *Post-Refactoring* 77.14**,** matrik *Cyclomatic Complexity* mendapatkan nilai rata rata *Pra-Refactoring* 11.2 dan *Post-Refactoring* 12.77*,* matrik *Depth of Inheritance* mendapatkan nilai rata rata *Pra-Refactoring* 1 dan *Post-Refactoring* ***1,*** matrik *Class Coupling* mendapatkan nilai rata rata *Pra-Refactoring* 7.25 dan *Post-Refactoring* 7.45 dan matrik *Lines of Source Code* mendapatkan nilai rata rata *Pra-Refactoring* 126.8 dan *Post-Refactoring* 94.48**.**

VI. SARAN

Dalam melakukan pembangunan dan pengembangan perangkat lunak, pemanfaatan konsep dari metode refactoring sangat membantu untuk mengurangi adanya bug secara berlebih dan mempermudah ketika akan melakukan penambahan fitur.

Adapun saran terhadap penelitian ini kedepannya dapat sebagai berikut :

1. Implementasi *refactoring* dapat dilakukan terhadap sisi front-end.
2. Melakukan *refactoring* dari tingkat *performance* dan berdampak bagi sisi eksternalnya.
3. Untuk penelitian selanjutnya dapat ditambahkan kohesi antar object untuk mengetahui nilai dari *refactoring* yang dilakukan.

REFERENSI

[1] R. Malhotra and K. Lata, “A systematic literature review on empirical studies towards prediction of software maintainability,” *Soft comput*, vol. 24, no. 21, pp. 16655–16677, 2020, doi: 10.1007/s00500-020-05005-4.

[2] M. S. Aen and A. Finandhita, “Penilaian Kualitas Perangkat Lunak Pada Aplikasi Akta Notaris Fidusia Di CV. Fredavelop,” *Digital library - Perpustakaan Pusat Unikom - Knowledge Center*, 2017.

[3] Skladannyi et al, “Modifikasi Model Pemeliharaan Delta dari Perangkat Lunak Berorientasi Objek dengan memberi peringkat dari 1 (tujuan,” vol. 6039, pp. 0–2, 2022.

[4] P. Hegedűs, I. Kádár, R. Ferenc, and T. Gyimóthy, “Empirical evaluation of software maintainability based on a manually validated refactoring dataset,” *Inf Softw Technol*, vol. 95, no. November 2017, pp. 313–327, 2018, doi: 10.1016/j.infsof.2017.11.012.

[5] S. Kaur and P. Singh, “How does object-oriented code refactoring influence software quality? Research landscape and challenges,” *Journal of Systems and Software*, vol. 157, 2019, doi: 10.1016/j.jss.2019.110394.

[6] I. Nikolov and C. Madsen, “Initial development of”Infection defender”: A children’s educational game for pandemic prevention measurements,” *VISIGRAPP 2021 - Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, vol. 1, no. Visigrapp, pp. 253–260, 2021, doi: 10.5220/0010284102530260.

[7] A. Almogahed, M. Omar, and N. H. Zakaria, “Impact of Software Refactoring on Software Quality in the Industrial Environment: A Review of Empirical Studies,” *in Knowledge Management International Conf. (KMICe) 2018*, no. July, pp. 25–27, 2018, [Online]. Available: http://www.kmice.cms.net.my/

[8] W. J. Mekel, S. R. A. Sompie, and B. A. Sugiarso, “Rancang Bangun Game 3D Pertahanan Kerajaan Bowontehu,” *Teknik Informatika*, vol. 14, no. 4, pp. 455–464, 2019.