TUGAS AKHIR – KI141502

# IMPLEMENTASI ALGORITMA HUNT-AND-KILL UNTUK PERANCANGAN PUZZLE PADA GAME 'PLANT THE FUTURE'

SYAUKI AULIA THAMRIN
NRP 5114100083

Dosen Pembimbing
Imam Kuswardayan, S.Kom, M.T.
Anny Yuniarti, S.Kom., M.Comp.Sc.

DEPARTEMEN INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018


**TUGAS AKHIR – KI141502**

# IMPLEMENTASI ALGORITMA HUNT-AND-KILL UNTUK PERANCANGAN PUZZLE PADA GAME 'PLANT THE FUTURE'

SYAUKI AULIA THAMRIN
NRP 5114100083

Dosen Pembimbing
Imam Kuswardayan, S.Kom, M.T.
Anny Yuniarti, S.Kom., M.Comp.Sc.

DEPARTEMEN INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018

*[Halaman ini sengaja dikosongkan]*


**FINAL PROJECT – KI141502**

# IMPLEMENTATION OF HUNT-AND-KILL ALGORITHM FOR PUZZLE ARRANGEMENTS ON 'PLANT THE FUTURE' GAME

SYAUKI AULIA THAMRIN
NRP 5114100083

Advisor
Imam Kuswardayan, S.Kom, M.T.
Anny Yuniarti, S.Kom., M.Comp.Sc.

INFORMATICS DEPARTEMENT
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018

*[Halaman ini sengaja dikosongkan]*

**LEMBAR PENGESAHAN**

# IMPLEMENTASI ALGORITMA HUNT-AND-KILL UNTUK PERANCANGAN PUZZLE PADA GAME 'PLANT THE FUTURE'

## TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Interaksi Grafika dan Seni
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :
**Syauki Aulia Thamrin**
NRP : 5114100083

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Imam Kuswardayan, S.Kom, M.T. .....
NIP: 197612152003121001 (pembimbing 1)

Amny Yuniarti, S.Kom., M.Comp.Sc. .....
NIP: 198106222005012002 (pembimbing 2)

**SURABAYA**
**JANUARI 2018**

*[Halaman ini sengaja dikosongkan]*

**IMPLEMENTASI ALGORITMA HUNT-AND-KILL UNTUK PERANCANGAN PUZZLE PADA GAME 'PLANT THE FUTURE'**


| | |
|---|---|
| Nama Mahasiswa | : SYAUKI AULIA THAMRIN |
| NRP | : 5114100083 |
| Departemen | : Informatika FTIK-ITS |
| Dosen Pembimbing I | : Imam Kuswardayan, S.Kom, M.T. |
| Dosen Pembimbing II | : Anny Yuniarti, S.Kom., M.Comp.Sc. |


## Abstrak


Kemajuan teknologi membuat banyak perubahan terutama dalam perkembangan game atau aplikasi permainan. Pengalaman bermain mulai berubah dari sesuatu yang statis menjadi dinamis, contohnya dalam penyusunan *maze* atau *puzzle* dalam aplikasi permainan. Cukup banyak algoritma yang dapat digunakan untuk menyusun *puzzle* dan membuatnya dinamis, namun *puzzle* sendiri memiliki aturan yang unik untuk masing-masingnya. Sehingga penerapan penyusunan *puzzle* perlu disesuaikan dengan *gameplay* atau aturan permainan yang ada pada game.

Pada penelitian ini penulis menawarkan sebuah cara penggunaan algoritma penyusunan *maze* yang dapat digunakan dan disesuaikan agar dapat menghasilkan *puzzle* yang berubah-ubah dan dapat diselesaikan. 'Plant the Future' merupakan game *puzzle* yang dirancang untuk *smartphone* berbasis Android. Game ini memiliki stage dengan susunan *puzzle* yang mirip dengan *maze* sederhana dengan aturan permainan yang cukup unik, yaitu dengan adanya dua *puzzle* pada setiap stage. *Puzzle* yang ada bervariasi dimensinya mulai dari 4x4 hingga 11x11. Dalam permainan ini pergerakan pemain sangat bergantung pada susunan *puzzle* untuk mendapatkan poin agar dapat memenangkan permainan. Oleh karena itu diperlukan sebuah cara agar susunan *puzzle* dapat dimenangkan. Untuk penyusunan *puzzle* maka digunakan algoritma Hunt-and-Kill agar dapat menghasilkan susunan *puzzle* yang bukan hanya dinamis, namun juga dapat diselesaikan.

Uji coba dilakukan dengan memainkan 30 stage dengan susunan *puzzle* yang dirancang menggunakan algoritma Hunt-and-Kill. Setiap stage diuji coba masih-masing sebanyak lima kali dimainkan. Setelah diuji coba maka diamati apakah susunan *puzzle* stage yang dihasilkan oleh algoritma Hunt-and-Kill dapat diselesaikan atau tidak. Pada hasilnya, algoritma Hunt-and-Kill dapat menyusun *puzzle* pada 30 stage game 'Plant the Future' dengan baik. *Puzzle* yang dirancang dengan dimensi bervariasi dari 4x4 sampai 11x11 dapat diselesaikan dan dimenangkan, sehingga algoritma ini dapat dikatakan cocok dengan aturan permainan game 'Plant the Future'.

***Kata kunci: Hunt-and-Kill, maze, puzzle***

**IMPLEMENTATION OF HUNT-AND-KILL ALGORITHM FOR PUZZLE ARRANGEMENTS ON PLANT THE FUTURE GAME**


| | |
|---|---|
| Name | : SYAUKI AULIA THAMRIN |
| NRP | : 5114100083 |
| Major | : Informatics – FTIK ITS |
| Supervisor I | : Imam Kuswardayan, S.Kom, M.T. |
| Supervisor II | : Anny Yuniarti, S.Kom., M.Comp.Sc. |


## Abstract


*Technological advances make many changes especially in the development of games or game applications. Play experience starts to change from something static to dynamic, for example in the preparation of mazes or puzzles in game applications. Quite a lot of algorithms can be used to construct puzzles and make them dynamic, but the puzzles themselves have unique rules for each. So the application of puzzle preparation needs to be adjusted with the gameplay or game rules that exist in the game.*

*In this study the authors offer a way of using maze compilation algorithms that can be used and customized in order to produce puzzles that change and can be solved. 'Plant the Future' is a puzzle game designed for Android-based smartphones. This game has a stage with a puzzle arrangement similar to a simple maze with a game rules that are quite unique. Puzzles vary in dimensions ranging from 4x4 to 11x11. In this game the movement of players is very dependent on the order of the puzzle to get points. Therefore we need a way to make the puzzle arrangement can be won. To arrange a puzzle, Hunt-and-Kill algorithm is used in order to generate an array of puzzles that are not only dynamic but also resolvable.*

*Trials are performed by playing 30 stages with a puzzle arrangement designed using the Hunt-and-Kill algorithm. Each stage is tested each stage five times played. Once tested, it is observed whether the order of puzzle stages generated by the Hunt-and-Kill algorithm can be solved or not. In the result, the Hunt-*

*and-Kill algorithm can construct the puzzle on 30 stage 'Plant the Future' games well. Puzzles designed with dimensions varying from 4x4 to 11x11 can be completed and won, so the algorithm can be said to match the game rules of 'Plant the Future' game.*

***Keywords:* Hunt-and-Kill, maze, puzzle**

# KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul "IMPLEMENTASI ALGORITMA HUNT-AND-KILL UNTUK PERANCANGAN *PUZZLE* PADA GAME 'PLANT THE FUTURE'".

Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan tugas akhir ini penulis bisa mendapatkan ilmu lebih serta memanfaatkan semua ilmu yang telah didapatkan pada saat berkuliah di Departemen Informatika FTIK ITS.

Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Orang tua, Kakak, Adik serta Saudara-saudara yang selalu mendoakan dan mendukung penulis.
3. Pak Imam Kuswardayan, S.Kom, M.T. selaku pembimbing I yang selalu memberikan arahan, motivasi dan bantuan sekaligus bimbingan kepada penulis selama pengerjaan Tugas Akhir.
4. Ibu Anny Yuniarti, S.Kom., M.Comp.Sc. selaku pembimbing II yang juga telah sangat membantu, dan membimbing saat pengerjaan Tugas Akhir ini.
5. Andreas Galang Anugerah dan Ade Nobi Miranto yang tergabung bersama penulis dalam tim fragments, yang telah membantu penulis selama pengerjaan Tugas Akhir.

6. Bapak dan Ibu Dosen Karyawan Teknik Informatika FTIf ITS yang telah memberikan ilmunya.
7. Teman-teman angkatan 2014 yang telah membantu, berbagi ilmu, menjaga kebersamaan, dan memberi motivasi kepada penulis, kakak-kakak angkatan 2013, 2012, serta adik-adik angkatan 2015 dan 2016 yang membuat penulis untuk selalu belajar.
8. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Januari 2018

# DAFTAR ISI

*[Halaman ini sengaja dikosongkan]*

# DAFTAR GAMBAR

# DAFTAR TABEL

*[Halaman ini sengaja dikosongkan]*

# DAFTAR KODE SUMBER

*[Halaman ini sengaja dikosongkan]*

# BAB I
# PENDAHULUAN

Bab ini membahas garis besar penyusunan tugas akhir yang meliputi latar belakang, tujuan pembuatan, rumusan dan batasan permasalahan, metodologi penyusunan tugas akhir, dan sistematika penulisan.

## 1.1. Latar Belakang

Aplikasi permainan menjadi salah satu hiburan untuk berbagai macam kalangan. Perkembangannya juga sangat cepat dan hal tersebut membuat aplikasi terus berkembang mengikuti zaman. Berbagai perkembangan teknologi juga diterapkan di aplikasi permainan dengan tujuan untuk menambah pengalaman bermain pemain. Salah satunya dalam penggunaan AI (Artificial Intelligence) untuk aplikasi permainan. Penggunan AI pada aplikasi permainan dapat menciptakan pengalaman bermain yang baru dan menarik, sebuah permainan digital yang adaptif secara emosional [1]. Pengalaman bermain mulai berubah dari sesuatu yang statis menjadi dinamis, contohnya dalam penyusunan *maze* atau *puzzle* dalam aplikasi permainan. Aplikasi permainan yang menjadikan *maze* sebagai inti permainannya contohnya adalah game 'Pac-Man'.

'Plant the Future' merupakan game *puzzle* yang dirancang untuk *smartphone* berbasis Android. Game ini memiliki stage berupa *puzzle* yang mirip dengan *maze* sederhana. Game Plant the Future sendiri memiliki aturan permainan yang cukup unik dengan memiliki dua tampilan *puzzle* dalam satu stagenya. Penyusunan stage tersebut perlu dibuat dinamis, sehingga diperlukan sebuah cara agar penyusunan *puzzle* stage ini dapat terus berubah-ubah dan tentunya dapat diselesaikan. Oleh karena itu untuk menyusun *puzzle* yang terus berubah-ubah maka digunakan algoritma Hunt-and-Kill untuk menghasilkan susunan *puzzle* yang bukan hanya dinamis, namun juga dapat diselesaikan.

Tujuan dari pengerjaan Tugas Akhir ini adalah mampu menghasilkan aplikasi permainan yang memiliki rancangan stage permainan berupa *puzzle* yang ditampilkan dalam bentuk *maze* sederhana dan akan terus berubah-ubah susunannya setiap kali dimainkan. *Puzzle* yang tercipta tentunya harus dapat dimenangkan. Selain itu, tugas akhir ini diharapkan mampu memberikan solusi terhadap penerapan Generate *Puzzle*, khususnya menggunakan metode Hunt-and-Kill, pada aplikasi permainan.

## 1.2. Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

a. Bagaimana merancang permainan ‘Plant the Future’?
b. Bagaimana mendapatkan aturan perancangan *puzzle* menggunakan algoritma Hunt-and-Kill?
c. Bagaimana menerapkan algoritma Hunt-and-Kill dalam merancang *puzzle* pada game mobile ‘Plant the Future’?

## 1.3. Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir memiliki beberapa batasan, yakni sebagai berikut:

a. Dimensi *puzzle* berupa *maze* sederhana dalam game ‘Plant the Future’ dibatasi hingga dimensi 11x11.
b. Algoritma Hunt-and-Kill yang digunakan disesuaikan dengan aturan permainan.
c. Algoritma Hunt-and-Kill dibuat dalam bahasa pemrograman C#.
d. Untuk interaksinya, permainan menggunakan sentuhan jari tangan untuk memainkan permainan sebagai input dalam aplikasi permainan.

## 1.4. Tujuan

Tujuan pembuatan tugas akhir ini antara lain:

a. Membuat aplikasi permainan berbasis mobile Android yang memiliki stage yang dinamis.
b. Pemanfaatan metode Hunt-and-Kill dalam merancang *puzzle* pada aplikasi permainan ‘Plant the Future’.

## 1.5. Manfaat

Manfaat dari penelitian tugas akhir ini adalah terciptanya aplikasi permainan khusunya *puzzle* dapat terus memberikan pemain pengalaman baru setiap kali memainkannya dengan menghasilkan *puzzle* yang berbeda setiap kali pemain memainkannya. Hasil penelitian dengan menggunakan algoritma Hunt-and-Kill ini diharapkan kedepannya mampu memberikan solusi terhadap penerapan algoritma Hunt-and-Kill untuk perancangan *puzzle* pada aplikasi permainan yang serupa ataupun yang lebih kompleks.

## 1.6. Metodologi Pembuatan Tugas Akhir

Pembuatan tugas akhir dilakukan menggunakan metodologi sebagai berikut:

A. Studi literatur

Tahap studi literatur merupakan tahap pembelajaran dan pengumpulan informasi yang digunakan untuk mengimplementasikan tugas akhir. Tahap ini diawali dengan pengumpulan literatur, diskusi, eksplorasi teknologi, dan pustaka, serta pemahaman dasar teori yang digunakan pada topik tugas akhir. Literatur-literatur yang dimaksud disebutkan sebagai berikut:

1. Unity
2. Bahasa pemrograman C#
3. Algoritma Hunt-and-Kill

B. Perancangan permainan

Pada tahap ini akan dilakukan analisa, perancangan, dan pendefinisian kebutuhan system untuk mengetahui permasalahan yang akan dihadapi pada tahap implementasi. Kemudian akan dijabarkan kebutuhan-kebutuhan tersebut

ke dalam perancangan fitur sistem. Berikut langkah yang akan dilakukan perancangan proses aplikasi:

1. Perancangan *gameplay*
2. Perancangan menu
3. Perancangan data dan *asset* permainan.
4. Algoritma Hunt-and-Kill dalam aplikasi permainan

C. Implementasi dan pembuatan sistem

Aplikasi ini dibangun menggunakan Game Engine Unity Personal Edition, dengan bahasa pemrograman C#.

D. Pengujian dan Evaluasi

Tahap pengujian dan evaluasi berisi pengujian aplikasi dan evaluasi berdasarkan hasil pengujian. Pada tahap ini dilakukan pengujian dari fungsionalitas perangkat lunak, apakah sesuai dengan yang diharapkan serta tidak diharapkan terdapat bug. Pengujian terhadap algoritma Hunt-and-Kill yang digunakan dilakukan dengan memastikan apakah stage dapat diselesaikan atau tidak.

E. Penyusunan laporan tugas akhir

Pada tahap ini dilakukan penyusunan laporan yang berisi dasar teori, dokumentasi dari perangkat lunak, dan hasil-hasil yang diperoleh selama pengerjaan tugas akhir.

## 1.7. Sistematika Penulisan

Buku Tugas Akhir ini terdiri atas beberapa bab yang tersusun secara sistematis, yaitu sebagai berikut.

1. BAB 1, Pendahuluan, menjelaskan latar belakang, batasan masalah, tujuan dari pembuatan tugas akhir ini serta metodologi yang digunakan selama penyusunan.
2. BAB 2, Tinjauan Pustaka, memaparkan hasil studi literatur yang digunakan sebagai dasar untuk menyelesaikan tugas akhir ini, terdiri atas deskripsi mengenai perancangan perangkat lunak, human computer interaction, Unity sebagai game engine, bahsa pemrograman C#, game *puzzle*, dan algoritma Hunt-and-Kill

3. BAB 3, Analisa dan Perancangan sistem game yang dikembangkan. Pada tahap ini dijelaskan deskripsi dari game Plant the Future dan dianalisa bagaimana *gameplay* dari game Plant the Future. Setelahnya dibahas menganai bagaimana perancangan *gameplay*, perancangan stage, dan bagaiman algoritma Hunt-and-Kill diterapakan dan disesuaikan dengan aturan permainan Plant the Future.
4. BAB 4, Bab ini membahas implementasi dari rancangan sistem yang dilakukan pada tahap perancangan. Penjelasan implementasi meliputi implementasi pembuatan aplikasi permain dengan menerapkan algoritma Hunt-and-Kill yang disesuaikan dengan aturan permainan.
5. BAB 5, Pengujian dan Evaluasi, pengujian dilakukan dengan menguji setiap stage yang menerapkan algoritma Hunt-and-Kill dalam penyusunan stagenya. Setiap stage akan diuji coba dan diamati apakah stage dapat diselesaikan atau tidak.
6. BAB 6, Kesimpulan dan Saran, berisi tentang kesimpulan yang didapat dari proses pembuatan tugas akhir beserta saran-saran untuk pengembangan selanjutnya.

*[Halaman ini sengaja dikosongkan]*

# BAB II
# TINJAUAN PUSTAKA

Bab ini membahas teori-teori yang mendukung pembuatan tugas akhir. Teori yang mendukung tersebut adalah deskripsi mengenai perancangan perangkat lunak, *human computer interaction*, Unity sebagai *game engine*, bahsa pemrograman C#, game *puzzle*, dan algoritma Hunt-and-Kill.

## 2.1. Human Computer Interaction

*Human Computer Interaction*, atau dalam bahasa Indonesia yaitu "Interaksi Manusia dan Komputer" merupakan ilmu yang mempelajari tentang interaksi antara manusia dan komputer. Merancang bagaimana sistem computer agar efektif, efisien, mudah dan menyenangkan agar masyarakat dapat menyadari manfaat perangkat komputasional [2].

## 2.2. Unity (Game Engine)

Unity merupakan sebuah game engine yang dikembangkan oleh Unity Technologies. Unity dapat menciptakan game ke dalam beberapa sistem operasi sekaligus. Antara lain: Windows Phone, Android, iOS, Windows 8, OSX, Tizen OS, Blackberry 10, Playstation 3, Playstation 4, XBOX, Oculus Rift dan sebagainya. Game yang dapat dibuat dengan Unity ini bisa dalam bentuk 3D atau 2D [3].

## 2.3. Bahasa Pemrograman C#

C# (dibaca: c sharp) merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari kerangka .Net framework. Bahasa pemrograman ini memiliki susunan yang mendekati bahasa C++.

## 2.4. Game *Puzzle*

*Puzzle* adalah salah satu genre permainan yang memiliki beberapa turunan contohnya *maze*. Game dengan genre *puzzle* sendiri memiliki banyak model permainan yang terus berkembang

mengikuti zaman. Game *puzzle* memiliki penyelesaian yang tak pasti dan tentunya menggunakan sejumlah waktu untuk menyelesaikannya [4].

## 2.5. Game Yang Mirip Dengan 'Plant the Future'

Terdapat game yang memiliki *gameplay* yang mirip dengan game 'Plant the Future' dari segi layout, tujuan permainan, dan pergerakan karakter, yaitu game Cute Munchies. Cute munchies adalah game puzzle yang berjalan di platform Android.


**Gambar 2.1 Game Cute Munchies**

Sumber : Game Cute Munchies

Tujuan pemain dalam permainan Cute Munchies adalah mendapatkan objek tertentu dalam stage. Karakternya akan bergerak sesuai arah *swipe* dan terus bergerak hingga menemui batas *puzzle* atau *obstacle.* Tempat yang telah dikunjungi oleh karakter akan jatuh sehingga karakter tidak dapat kembali lagi ke posisi sebelumnya.

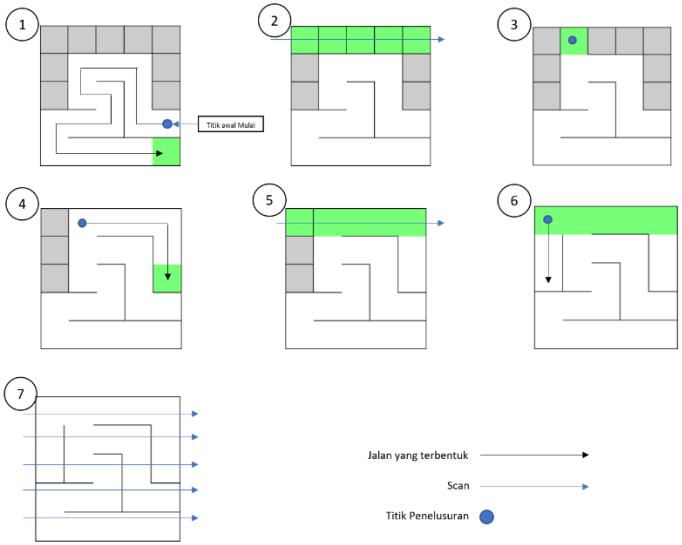
## 2.6. Algoritma Hunt-and-Kill

Hunt-and-Kill adalah salah satu algoritma yang dapat digunakan untuk menghasilkan *maze*. Algoritma Hunt-and-Kill bertujuan untuk memperoleh susunan *maze* secara otomatis dan

dapat diselesaikan. Dalam bukunya, Jamis Buck berpendapat bahwa algoritma Hunt-and-Kill ini mirip dengan algoritma Aldous-Broder.

Hunt-and-Kill dan Aldous-Broder memiliki perbedaan dalam sistem penelusuran *maze*. Aldous-Broder menelusuri dengan langkah acak bahkan ke tempat yang sudah pernah dilalui sedangkan Hunt-and-Kill hanya bergerak ke tempat yang belum dilalui saja. Langkah-langkah berjalannya algoritma Hunt-and-Kill adalah sebagai berikut [5]:

1. Tahap pertama adalah memilih titik awal inisialisasi secara acak.
2. Tahap ‘Kill’ Dimulai dari titik awal, secara bebas memilih arah yang akan dituju (atas, kanan, bawah, kiri) selama tempat yang akan dituju belum belum pernah dilalui dan tandai tempat tersebut sebagai tempat yang dapat dilalui. Penelusuran akan terus dilanjutkan hingga menemui jalan buntu. Jika menemui jalan buntu maka akan masuk ke tahap ‘Hunt’.
3. Tahap ‘Hunt’ dimulai dari posisi atas, akan melakukan scan dari kiri ke kanan. Scan akan dilakukan hingga menemui tempat yang belum pernah dikunjungi dan tempat tersebut bersebelahan dengan tempat yang sudah pernah dilalui, seperti terlihat pada Gambar 2.2. nomor 2 dan 3. Jika tempat yang dicari berhasil didapatkan maka kembali ke tahap ‘Kill’ dengan menggunakan titik yang didapatkan sebagai titik awal. Penelusuran akan terus dilakukan seperti terlihat di Gambar 2.2 nomor 4 hingga 6 hingga seluruh tempat telah semuanya di kunjungi. Setelah semua tempat dikunjungi maka algoritma berhenti dan didapatkan susunan *maze* seperti yang terlihat di Gambar 2.2 nomor 7.




**Gambar 2.2 Algoritma Hunt-and-Kill**

# BAB III
# ANALISIS DAN PERANCANGAN

Bab ini membahas tahap analisis permasalahan dan perancangan tugas akhir. Pada bagian awal akan dibahas mengenai *gameplay* atau aturan dari game 'Plant the Future'. Selajutnya dibahas mengenai penggunaan algoritma Hunt-and-Kill yang digunakan untuk menghasilkan stage yang dinamis. Algoritma Hunt-and-Kill yang digunakan pada game Plant the Future telah disesuaikan dengan aturan permainan agar dapat membuat stage dinamis yang dapat diselesaikan.

## 3.1. Analisis

### 3.1.1. Deskripsi Game Plant the Future

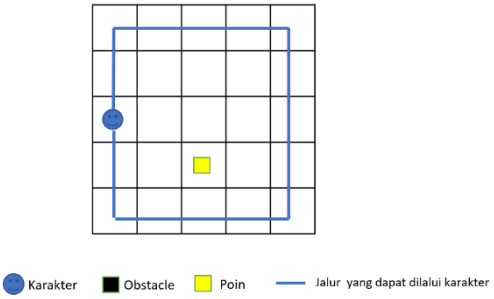
Plant the Future yang dalam artian bahasa indonesia adalah menanam masa depan adalah sebuah game bergenre *puzzle* dengan tema edukasi lingkungan. Pada game ini pemain dihadapkan pada sebuah *puzzle* yang berbentuk *maze* kecil. Menariknya dalam game ini, pemain dapat mengakses dua *maze* atau *puzzle* secara bergantian. Pemain dapat menyelesaikan stage dengan mengumpulkan poin, poin dikumpulkan dengan cara menelusuri *maze* dan melakukan perpindahan waktu.

### 3.1.2. Analisis *Gameplay* Aturan Permainan

Plant the Future memiliki aturan yang cukup sederhana. Untuk awal mula permainan pemain akan dihadapakan pada *puzzle* stage. *Puzzle* yang berbentuk seperti *maze* kecil dalam permainan ini ada dua, yaitu *puzzle* 'masa depan' dan *puzzle* 'masa lalu'. Pemain dapat mengakses keduanya secara bergantian dengan cara melakukan perpindahan waktu.

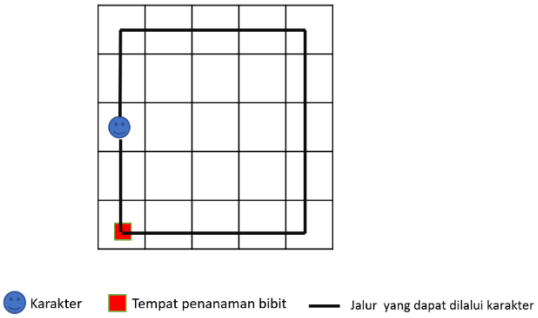
Pemain memiliki sebuah karakter yang dapat digerakkan. Pemain dapat menggerakkannya dengan melakukan *swipe* pada layar permainan. Saat melakukan *swipe* karakter pemain akan terus bergerak dan berhenti ketika bertemu dengan batas *puzzle* atau bertemu dengan *obstacle*.

Dalam game Plant the Future pemain harus mendapatkan seluruh poin yang ada dalam *puzzle* untuk memenangkan permainan poin. Poin hanya berada di ‘masa depan’. Namun beberapa poin tidak bisa didapatkan karena tidak adanya tempat pemberhentian seperti terlihat di Gambar 3.1. Untuk membuat pemain dapat mencapai poin maka diperlukan sebuah cara, yaitu dengan melakukan perpindahan waktu.



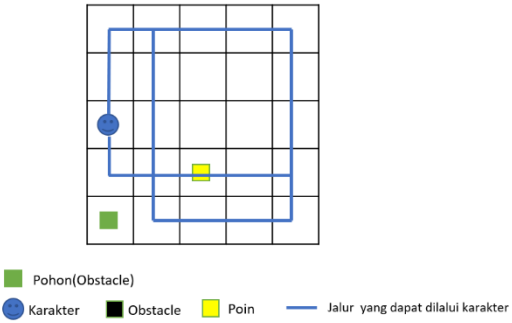

**Gambar 3.1 Poin yang tidak dapat dicapai di ‘Masa Depan’**

Pemain dapat pindah ke masa lalu dengan memilih tombol *change time*. Di ‘masa lalu’ terdapat objek tempat penanaman bibit, terlihat di Gambar 3.2. Pemain dapat menanam bibit dengan cara melewati atau berhenti di tempat penanaman bibit. Saat pemain berhasil menanam bibit maka bibit akan tumbuh di posisi yang sama di masa depan dan menjadi pohon(*obstacle*).




**Gambar 3.2 Tempat penanaman bibit di 'Masa Lalu'**

Setelah pemain kembali ke ’masa depan’ dan pohon telah tumbuh. Pemain kini memiliki tempat pemberhentian yang lebih banyak dan bisa mendapatkan poin yang ingin dicapai seperti terlihat di Gambar 3.3. Pemain dapat menggunakan konsep yang sama untuk mendapatkan seluruh poin untuk memenangkan permainan.




**Gambar 3.3 Poin bisa didapatkan dengan adanya pohon**

Pemain dapat mendapatkan pengalaman memainkan *puzzle* yang menarik dengan mengkombinasikan fitur-fitur yang ada dalam permainan seperti perpindahan waktu dan penanaman bibit. Dalam tugas akhir ini penulis juga mengatur agar susunan *puzzle* berubah setiap kali pemain memainkannya dengan menggunakan algoritma Hunt-and-Kill sebagai penyusun *puzzle*.

### 3.1.3. Analisis Komponen Permainan

Ada beberapa komponen yang ada dalam aplikasi permainan ‘Plant the Future’. Komponen tersebut akan mempengaruhi jalannya permainan.

#### 3.1.3.1. Susunan *Puzzle*

Plant the Future memiliki susunan stage berupa *puzzle* yang ditampilkan dalam bentuk *maze* kecil. *Puzzle* yang ada pada game ini memiliki dimensi yang berbeda tergantung dari stage yang dimainkan. Pada game ini pemain dapat mendapatkan susunan *puzzle* yang berbeda setiap kali memainkan stage. Susunan yang berbeda didapatkan dari penggunaan algoritma Hunt-and-Kill dalam penyusunan *puzzle*. Game ini memiliki dua *puzzle*, yaitu *puzzle* masa depan dan *puzzle* masa lalu. Kedua *puzzle* dapat diakses secara bergantian dengan menggunakan tombol perpindahan waktu dalam tampilan permainan.

#### 3.1.3.2. *Obstacle*

*Obstacle* adalah objek dalam permainan. *Obstacle* pada game Plant the Future berupa batu dan pohon. *Obstacle* bertindak sebagai penghalang untuk menghambat pergerakan karakter pemain. Walaupun bertindak sebagai penghalang obstacle dapat membantu pemain untuk mendapatkan poin dan menyelesaikan permainan, karena obstacle dapat memberikan tempat pemberhentian karakter yang lebih banyak kepada pemain. Sehingga pemain dapat memanfaatkannya. Jumlah *obstacle* pada sebuah stage ditentukan oleh Algoritma Hunt-and-Kill yang digunakan untuk menyusun *puzzle*.

#### 3.1.3.3. Poin

Poin adalah objek yang harus diambil pemain untuk memenangkan permainan. Jumlah poin harapan untuk tiap stage diatur sedemikian rupa. Untuk detail lebih lengkapnya untuk pembagian jumlah poin harapan yang ada pada stage dapat dilihat pada bagian 3.2.1.

#### 3.1.3.4. Tempat Penanaman Bibit

Tempat penanaman bibit adalah objek yang ada dalam stage. Objek ini hanya ada pada *maze* masa lalu. Pemain dapat menanam bibit di tempat penanaman bibit ini. Setelah bibit ditanam maka pada *maze* masa depan akan ada pohon yang tumbuh dan dapat dianggap sebagai *obstacle*.

## 3.2. Perancangan

### 3.2.1. Perancangan Stage

Game Plant the Future memiliki 31 stage. Terdapat stage awal sebagai stage tutorial dan 30 stage yang menerapkan algoritma Plant the Future. Setiap stage memilik dimensi yang berbeda mulai dari 3x3, 4x4, 5x4, 5x5 dan seterusnya hingga 11x11. Jumlah poin yang ada di stage juga berbeda. Untuk 30 stage yang menerapkan Hunt-and-Kill memiliki waktu penyelesaian stage(batas waktu pemain menyelesaikan stage). Berikut pembagian format stage pada tiap stage:

**Tabel 3.1 Konfigurasi stage game**

| No | Stage | Jumlah Poin | Dimensi | Batas Waktu Penyelesaian |
|---|---|---|---|---|
| **1** | 1 | 1 | 3x3 | - |
| **2** | 2 | 2 | 4x4 | 30s |
| **3** | 3-1 | 3 | 5x4 | 35s |
| **4** | 3-2 | 3 | 5x5 | 35s |
| **5** | 4-1 | 4 | 6x4 | 40s |
| **6** | 4-2 | 4 | 6x5 | 40s |
| **7** | 4-3 | 4 | 6x6 | 40s |
| **8** | 5-1 | 5 | 7x4 | 50s |
| **9** | 5-2 | 5 | 7x5 | 50s |
| **10** | 5-3 | 5 | 7x6 | 50s |
| **11** | 5-4 | 5 | 7x7 | 50s |
| **12** | 6-1 | 6 | 8x4 | 60s |
| **13** | 6-2 | 6 | 8x5 | 60s |
| **14** | 6-3 | 6 | 8x6 | 60s |
| **15** | 6-4 | 6 | 8x7 | 60s |
| **16** | 6-5 | 6 | 8x8 | 60s |

| | | | | |
|---|---|---|---|---|
| **17** | 7-1 | 7 | 9x5 | 70s |
| **18** | 7-2 | 7 | 9x6 | 70s |
| **19** | 7-3 | 7 | 9x7 | 70s |
| **20** | 7-4 | 7 | 9x8 | 70s |
| **21** | 7-5 | 7 | 9x9 | 70s |
| **22** | 8-1 | 8 | 10x6 | 80s |
| **23** | 8-2 | 8 | 10x7 | 80s |
| **24** | 8-3 | 8 | 10x8 | 80s |
| **25** | 8-4 | 8 | 10x9 | 80s |
| **26** | 8-5 | 8 | 10x10 | 80s |
| **27** | 9-1 | 9 | 11x7 | 90s |
| **28** | 9-2 | 9 | 11x8 | 90s |
| **29** | 9-3 | 9 | 11x9 | 90s |
| **30** | 9-4 | 9 | 11x10 | 90s |
| **31** | 9-5 | 9 | 11x11 | 90s |

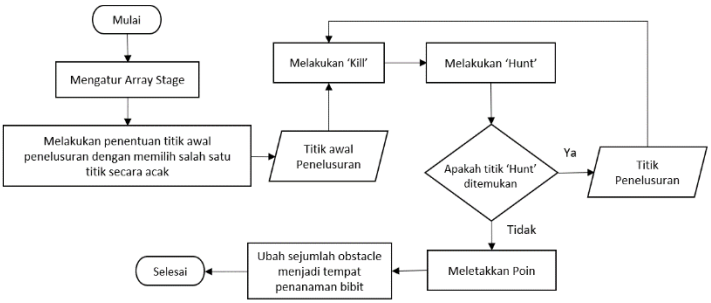
### 3.2.2. Perancangan Asset

Semua asset gambar merupakan asset internal penulis yang dibuat oleh pihak kedua dan sudah diberi izin pemakaian asset. Asset suara didapatkan oleh penulis dari audioblocks.com melalui akun Lab IGS.

### 3.2.3. Perancangan Algoritma Hunt-and-Kill

Algoritma Hunt-and-Kill diterapkan pada *gameplay* game Plant the Future pada bagian penyusunan *puzzle* stage. Algoritma Hunt-and-Kill yang diterapkan memiliki beberapa perubahan. Meskipun terjadi beberapa perubahan, inti dari algoritmanya sendiri tidak berubah. Beberapa perubahan tersebut dirubah agar dapat menyesuaikan dengan *gameplay* Plant the Future.

Penyesuaian terhadap algoritma Hunt-and-Kill terdapat pada kondisi saat proses penyusunan *maze*. Hunt-and-Kill yang digunakan sebagai penyusun *puzzle* pada game Plant the Future memiliki bagian yang sama dengan algoritma Hunt-and-Kill biasa. Algoritma dibagi menjadi dua mode atau tahap yaitu mode ‘Hunt’ dan mode ‘Kill’. Gambaran dari bagaimana jalannya dua bagian tersebut pada jalannya algoritma digambarkan pada flow diagram pada Gambar 3.4.




**Gambar 3.4 Flow diagram Algoritma Hunt-and-Kill**

Berdasarkan gambar tersebut, berikut adalah tahapan perancangan dan penerapan algoritma Hunt-and-Kill pada aplikasi permainan:

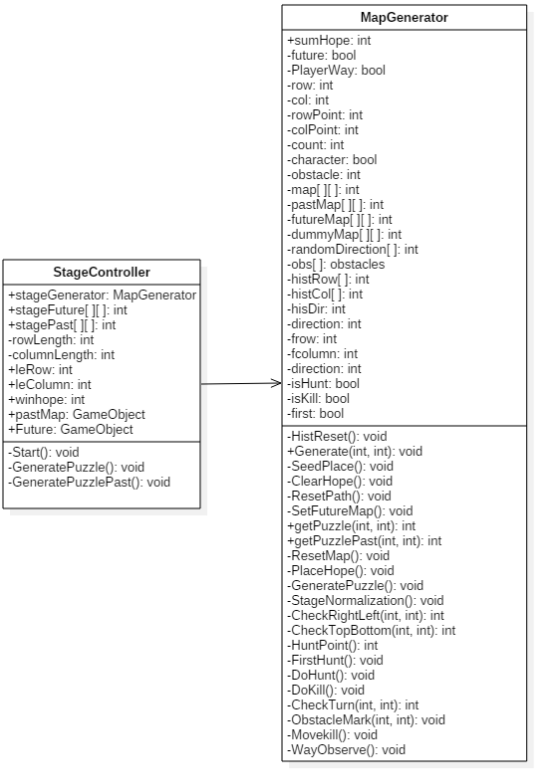
1. Jalannya algoritma Hunt-and-Kill dimulai dari terlebih dahulu mengatur array yang dibutuhkan. Array yang diterapkan untuk *puzzle* merupakan array dua dimensi.
2. Setelah array dibentuk ambil titik awal penelusuran terlebih dahulu. Titik awal penelusuran akan dipilih secara random dari array yang ada. Titik awal juga merupakan titik awal posisi karakter pemain. Setelah titik awal penelusuran didapatkan maka selanjutnya dimulailah tahap 'Kill'.
3. Saat tahap 'Kill' penelusuran akan dilakukan dimulai dari titik awal penelusuran. Dari titik awal penelusuran, penelusuran dilakukan dengan cara memilih secara acak(*random*) dari empat arah(atas, kanan, bawah, kiri) yang dapat dilalui dan belum di lakukan pengecekan. Setelah arah didapatkan maka titik penelusuran berpindah ke titik yang dimaksudkan. Tahapan 'kill' terus diulangi hingga penelusuran tidak menemukan potensi jalan yang dapat ditelusuri. Penjelasan lebih

detail tentang ‘Kill’ terdapat di bagian 3.2.3.2. Jika kill selesai maka masuk ke mode ‘Hunt’.

4. Mode ‘hunt’ yang akan dimulai dengan melakukan pengecekan dari titik [0,0]. Pengecekan untuk menemukan titik hunt akan berhenti saat menemukan titik hunt yang sesuai dengan persyaratan. Persyaratan inilah yang membuat algoritma Hunt-and-Kill yang digunakan dalam game Plant the Future berbeda dengan Hunt-and-Kill biasa. Penjelasan lebih detail mengenai ‘Hunt’ dan persyaratannya terdapat pada bagian 3.2.3.1.
5. Jika persyaratan terpenuhi maka didapatkan titik awal penelusuran. ‘Kill’ akan dimulai dari titik yang telah didapatkan tersebut. ‘Kill’ untuk melakukan penelusuran ini dilakukan secara berulang hingga mengalami *dead end,* yaitu tidak ditemukan lagi potensi titik ‘Kill’.
6. Jika tidak ada lagi titik ‘Hunt’ yang ditemukan maka letakkan poin sesuai jumlah yang ditentukan. Poin diletakkan secara random pada titik yang dapat dilewati karakter.
7. Setelah poin diletakkan maka ubah sejumlah obstacle(100% jika jumlah obstacle kurang dari 2 dan 75% jika jumlah obstacle lebih dari 2) menjadi tempat penanaman bibit. Obstacle diubah dengan cara memilihnya secara random dan diubah nilai arraynya menjadi tempat penanaman bibit secara acak.

Berbeda dengan Hunt-and-Kill biasa, di Hunt-and-Kill yang digunakan dalam ‘Plant the Future’ bisa jadi ada titik yang tidak dicek. Titik yang tidak di cek tersebut menandakan bahwa titik tersebut tidak dapat dilewati oleh pemain(karakter tidak memiliki akses untuk melewati titik tersebut). Hal tersebut dikarenakan adanya persyaratan terhadap titik hunt. Namun, titik yang tidak di cek tersebut akan dianggap sebagai objek jalan biasa saat dilakukan realisasi terhadap tampilan *gameplay*.

Penerapan Hunt-and-Kill dalam perancangan aplikasi permainan digambarkan dalam Class Diagram. Gambaran class diagramnya terdapat pada Gambar 3.5.

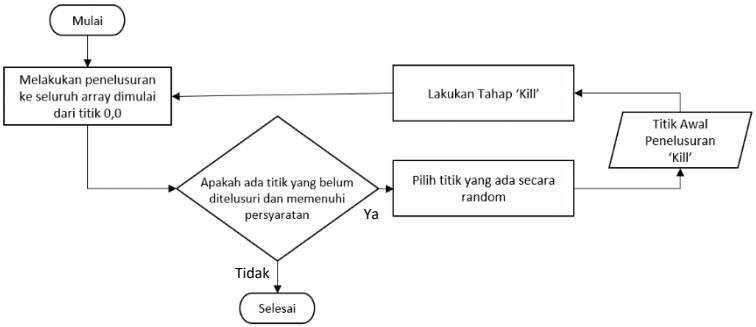



**Gambar 3.5 Class diagram rancangan Hunt-and-Kill**

Hunt-and-Kill akan menghasilkan susunan *puzzle* dalam bentuk array dua dimensi. Array yang didapatkan kemudian akan direalisasikan menjadi *gameplay* permainan dalam bentuk tampilan *puzzle*. Realisasi array akan lebih dijelaskankan bada bagian perancangan realisasi tampilan di bagian 3.2.4. Array akan digunakan untuk menampilkan tampilan *puzzle* masa depan dan *puzzle* masa lalu.

### 3.2.3.1. Hunt

Hunt bertujuan untuk mendapatkan titik awal penelusuran setiap iterasi. ‘Hunt’ terdiri dari beberapa tahapan. Tahapan tersebut digambarkan dalam flow diagram yang dapat dilihat pada Gambar 3.6.




**Gambar 3.6 Flow diagram tahap ‘Hunt’**

Berdasarkan Gambar 3.10, berikut adalah tahapan ‘Hunt’dari bagian algoritma Hunt-and-Kill yang diterapkan pada aplikasi permainan:

1. ‘Hunt’ dimulai dengan melakukan pencarian titik awal penelusuran dari titik 0,0 yang dilakukan secara horizontal hingga ujung array, kemudian berpindah baris dan melakukan pencarian lagi secara horizontal. Hal ini dilakukan terus hingga menemukan titik awal penelusuran atau jika tidak ada lagi titik yang dapat di cek.
2. Titik penelusuran didapatkan jika titik tersebut memenuhi beberapa kualifikasi atau syarat tertentu. Syarat titik dapat dijadikan titik awal penelusuran adalah sebagai berikut:

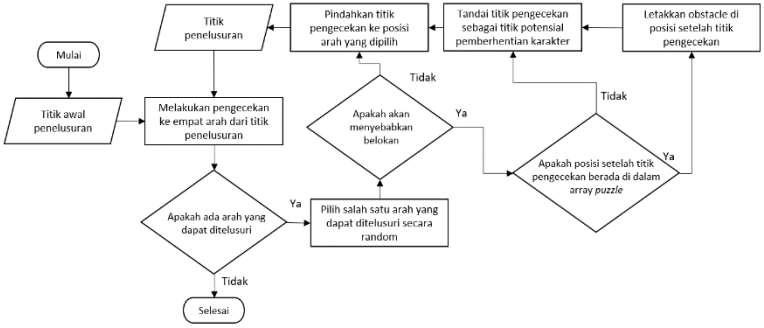
- Titik bersebelahan dengan posisi awal karakter pemain (titik yang ditemukan saat tahap ‘Hunt awal’).
- Titik bersebelahan dengan titik potensial pemberhentian karakter (titik dimana karakter pemain dapat berhenti, dijelaskan pada tahap kill pada bagian 3.2.3.2).

3. Jika titik ditemukan maka titik tersebut akan menjadi titik awal penelusuran ‘kill’, dan tahap ‘Kill’ dimulai.
4. Jika hingga batas pencarian tidak ditemukan titik yang potensial maka ‘hunt’ berhenti dan algoritma selesai dijalankan. Jalannya algoritma ‘Hunt’ dijelaskan dalam pseudocode sebagai berikut:

```
GeneratePuzzle()   {
     while isHunt == true then
             DoHunt();
}

DoHunt()  {
     if first == true then
             FirstHunt();
     else
             huntStatus = HuntPoint();

     if huntStatus == 0  then
             isHunt = false;

     while    isKill  ==  true then
             DoKill();

}
```

### 3.2.3.2. Kill

'Kill' bertujuan untuk melakukan penelusuran pembentukan susunan *puzzle* dari titik awal yang didapatkan dari proses 'Hunt'. 'Kill' terdiri dari beberapa tahapan. Tahapan tersebut digambarkan pada Gambar 3.7.




**Gambar 3.7 Flow diagram tahap 'Kill'**

Berdasarkan Gambar 3.7, berikut adalah tahapan 'Kill' dari bagian algoritma Hunt-and-Kill yang diterapkan pada aplikasi permainan:

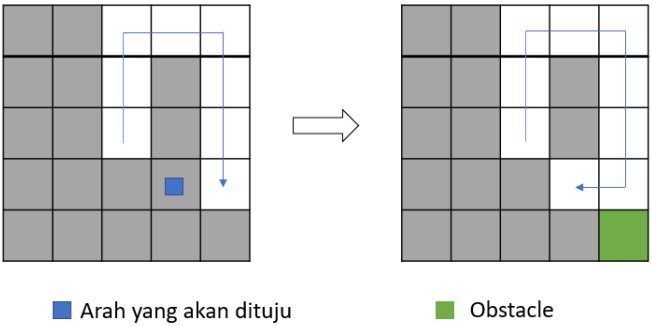
1. Tahap 'Kill' dimulai dari titik penelusuran yang didapatkan dari tahap 'Hunt' awal atau 'Hunt'
2. Dari titik penelusuran, dilakukan pengecekan ke empat arah yaitu atas, kanan, bawah, kiri. Syarat arah dapat ditelusuri adalah:
   - Titik atau posisi yang ditunjukkan arah masih belum pernah di cek sebelumnya.
   - Titik atau posisi yang ditunjukkan arah bukan merupakan *obstacle.*
   - Titik atau posisi yang ditunjukkan arah menyebabkan belokan namun masih memenuhi syarat terjadinya belokan (belokan dapat terjadi

jika posisi setelah titik pengecekan masih belum di cek atau titik pengecekan berada di batas dimensi *puzzle*).

3. Jika telah melakukan pengecekan ke empat arah dan mendapatkah hasil dari pengecekan ada tidaknya arah yang dapat ditelusuri maka akan terdapat beberapa kondisi, yaitu:
    - Jika tidak ada arah yang dapat ditelusuri maka berarti *dead end* yang berarti tahap 'Kill' menemukan jalan buntu dan tahap 'Kill' dinyatakan selesai, setelah 'Kill' selesai maka algoritma akan kembali melakukan 'Hunt'.
    - Jika ada arah yang dapat ditelusuri maka pilih salah satu arah secara acak dan lanjut ke step empat.
4. Jika arah ada dan sudah dipilih secara acak maka dilakukan pengecekan apakah arah yang dipilih akan menyebabkan belokan atau tidak, pengecekan akan membuat beberapa kondisi, sebagai berikut:
    - Jika arah tidak menyebabkan belokan maka langsung menuju step tujuh.
    - Jika arah menyebabkan belokan maka jalannya algoritma akan lanjut ke step lima.
5. Jika arah yang dipilih menyebabkan belokan maka dilakukan pengecekan terkait apakah titik setelah titik pengecekan berada di luar array *puzzle*. Pengecekan akan menyebabkan beberapa kondisi, yaitu sebagai berikut:
    - Jika posisi titik setelah titik pengecekan berada di luar array *puzzle* (contoh: jika *puzzle* berdimensi 5x5 dan penghitungan array dimulai dari [0,0] dengan situasi titik pengecekan berada di [3,4], menyebabkan belokan ke [2,4]. Maka titik setelah titik pengecekan adalah titik di posisi [3,5] yang

berarti berada di luar array *puzzle*) maka langsung menuju step enam.

- Jika posisi titik setelah titik pengecekan berada di dalam (tidak diluar array *puzzle*) maka letakkan *obstacle* dengan cara menandai posisi setelah titik pengecekan sebagai *obstacle*. Kemudian lanjut ke step enam. Peletakan obstacle dapat dilihat pada Gambar 3.8.




**Gambar 3.8 Peletakan *obstacle***

6. Tandai posisi pengecekan sebagai titik potensial pemberhentian karakter.
7. Pindahkan titik pengecekan ke posisi sesuai arah yang dipilih sesbelumnya dan jadikan titik tersebut sebagai titik penelusuran yang baru. Setelah didapatkan titik penelusuran yang baru maka kembali ke step dua untuk melakukan iterasi berikutnya.

'Kill' akan dilakukan terus secara berulang hingga tidak ada lagi arah yang dapat dilalui dari titik pengecekan. Setelah tidak ada lagi arah yang dapat dilalui maka 'Kill' selesai. Jalannya algoritma akan kembali ke tahap 'Hunt'. Pseudocode jalannya tahap 'Kill' adalah sebagai berikut:

```
DoKill(){
        WayObserve();
        // mengetahui arah yang dapat dilalui
```

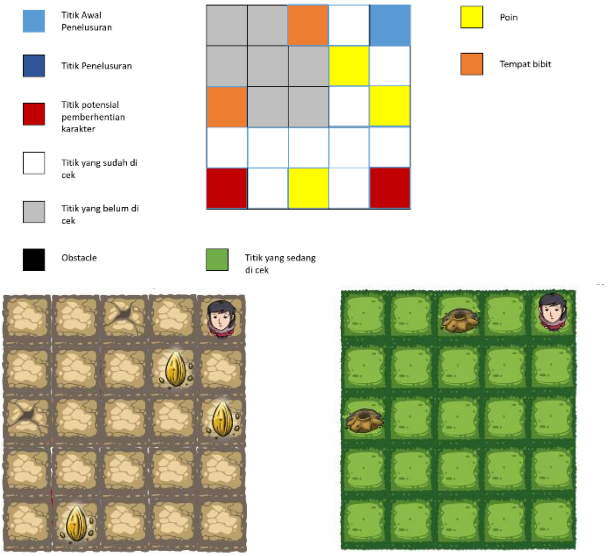
```
            if character == true then
                // letakkan karakter
                map[rowPoint, columnPoint] = karakter
                character = false
            else
                map[rowPoint, columnPoint] = jalan

            if  count <= 0
                // tidak terdapat arah yang bisa dilalui
                isKill = false

            else if count > 0 then
                // terdapat arah yang bisa dilalui
                direction = randomDirection
                Movekill()

        }

    MoveKill(){
            // cek belokan
            if CheckTurn() == true then
                 // cek apakah titik berada di batas puzzle
                  if rowPoint!= y-1 || colPoint != x-1 then
                    // meletakkan obstacle di titik setelah
                    titik pengecekan
                    map[] + 1 = obstacle
                    map[] = titik_berhenti

            // titik pengecekan berpindah sesuai arah
            rowPoint = newRowPoint
            colPoint = newColPoint
        }
```

### 3.2.4. Perancangan Realisasi Tampilan Permainan

Tampilan permainan direalisasikan berdasarkan array puzzle yang didapatkan dari jalannya algoritma Hunt-and-Kill. Untuk tampilan *ground* dan *background* permainan akan ada di kedua puzzle di ‘masa depan’ ataupun di ‘masa lalu’. Namun untuk tampilannya sendiri berbeda, sehingga pemain dapat mengetahui dia sedang berada di puzzle mana(‘masa depan’ atau ‘masa lalu’). Seluruh susunan puzzle awalnya akan direalisasikan menjadi *ground* termasuk tempat pemberhentian karakter. Setelah

itu barulah objek lainnya disusun berdasarkan susunan puzzle yang dihasilkan algoritma Hunt-and-Kill.

Beberapa objek hanya ada di ‘masa lalu’ atau hanya ada di ‘masa depan’. Untuk poin hanya akan ditampilkan di ‘masa depan’. Sedangkan untuk tempat penanaman bibit hanya akan ditampilkan di ‘masa lalu’ tapi pemain akan melihat tanda berupa retakan di posisi yang sama (tempat penanaman bibit) di ‘masa depan. Untuk *obstacle* akan direalisasikan sama di kedua puzzle namun dengan tampilan yang berbeda. Contoh hasil realisasi susunan array menjadi tampilan puzzle dapat dilihat pada Gambar 3.9.




**Gambar 3.9 Array hasil(atas), tampilan realisasi'masa depan'(kiri) dan 'masa lalu'(kanan)**

# BAB IV
# IMPLEMENTASI

Bab ini membahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Implementasi yang dijelaskan adalah bagaimana menerapkan algoritma Hunt-and-Kill dalam game Plant the Future untuk menghasilkan stage yang dinamis. Game Engine yang digunakan adalah Unity dengan bahasa pemrograman C#.

## 4.1. Lingkungan Pengembangan Sistem

Lingkungan pengembangan sistem yang digunakan untuk mengembangkan Tugas Akhir ini dilakukan pada lingkungan pengembangan sebagai berikut.

1. Sistem operasi Windows 10 Education 64 bit.
2. Unity Game Engine 2017
3. Visual Studio 2017 sebagai editor

## 4.2. Implementasi *Gameplay*

Implementasi dari semua fungsi diwujudkan dalam bentuk code dengan Bahasa pemrograman C#.

### 4.2.1. Implementasi Pergerakan Pemain

Pada saat bermain, pemain dapat menggerakkan karakter pemain dengan melakukan *swipe* pada layar permainan. Karakter pemain akan terus bergerak hingga menemukan batas *maze* atau menemui *obstacle*. Pemain tidak dapat berbelok saat karakter sedang bergerak. Implementasi pergerakan karakter oleh pemain sebagaimana yang telah dijelaskan dapat dilihat pada Kode Sumber 4.1.

```
1.  void  Update ()  {
2.      if  (!isMove)         {
3.          if  (!isTouch  &&  Input.GetMouseButtonDown(0))             {

4.              positionInit  =  Input.mousePosition;
5.              isTouch  =  true;
6.          }
```

```
7.          else  if  (Input.GetMouseButtonUp(0))             {

8.              isMove  =  true;
9.              swipeDirection();
10.             isTouch  =  false;
11.         }
12.         else  if  (isTouch)             {
13.             positionFinal  =  Input.mousePosition;
14.         }
15.     }
16. }
17. void  swipeDirection()     {
18.     isSwipe  =  false;
19.     GameObject  bgMusic  =  GameObject.FindGameObjectWithTag("BGM");

20.     float  inputX  =  positionFinal.x  -  positionInit.x;
21.     float  inputY  =  positionFinal.y  -  positionInit.y;
22.     float  slope  =  inputY  /  inputX;
23.     int  fSensitivity  =  15;
24.     float  distance  =  Mathf.Sqrt(Mathf.Pow((positionFinal.y  -  posit
    ionInit.y),  2)  +  Mathf.Pow((positionFinal.x  -  positionInit.x),  2)
    );
25.     if  (distance  <=  (Screen.width  /  fSensitivity))         {

26.         bgMusic.GetComponents < AudioSource > ()[1].enabled  =  false;

27.         isMove  =  false;
28.         return;
29.     }
30.     if  (!isGuide)  {
31.         if  (inputX  >=  0  &&  inputY  >  0  &&  slope  >  1)
        {
32.             StartCoroutine(move_up());
33.         }
34.         else  if  (inputX  <=  0  &&  inputY  >  0  &&  slope  <  -
    1)             {
35.             StartCoroutine(move_up());
36.         }
37.         else  if  (inputX  >  0  &&  inputY  >=  0  &&  slope  <  1  &&
      slope  >=  0)             {
38.             StartCoroutine(move_right());
39.         }
```

```
40.         else if (inputX > 0 && inputY <= 0 && slope > -
    1 && slope <= 0)             {
41.             StartCoroutine(move_right());
42.         }
43.         else if (inputX < 0 && inputY >= 0 && slope > -
    1 && slope <= 0)             {
44.             StartCoroutine(move_left());
45.         }
46.         else if (inputX < 0 && inputY <= 0 && slope >= 0 &
    & slope < 1)             {
47.             StartCoroutine(move_left());
48.         }
49.         else if (inputX >= 0 && inputY < 0 && slope < -
    1)             {
50.             StartCoroutine(move_down());
51.         }
52.         else if (inputX <= 0 && inputY < 0 && slope > 1)
              {
53.             StartCoroutine(move_down());
54.         }
55.     }
56.     else if (isGuide)         {
57.         if (stateGuide == 4)             {
58.             if ((inputX >= 0 && inputY > 0 && slope > 1) ||
     (inputX <= 0 && inputY > 0 && slope < -
    1))                 {
59.                 if (PlayerPrefs.GetInt("soundEffect") == 1)
                    bgMusic.GetComponents < AudioSource > ()[1].enabled =
     true;
60.                 StartCoroutine(move_up());
61.                 stateGuide = 0;
62.             }
63.             else isMove = false;
64.         }
65.         else if (stateGuide == 1)             {

66.             if ((inputX > 0 && inputY >= 0 && slope < 1 &&
     slope >= 0) || (inputX > 0 && inputY <= 0 && slope > -
    1 && slope <= 0))                 {
67.                 if (PlayerPrefs.GetInt("soundEffect") == 1)
                    bgMusic.GetComponents < AudioSource > ()[1].enabled =
     true;
68.                 StartCoroutine(move_right());
```

```
69.                 stateGuide  =  0;
70.             }
71.             else  isMove  =  false;
72.         }
73.         else  if  (stateGuide  ==  3)             {
74.             if  ((inputX  <  0  &&  inputY  >=  0  &&  slope  >  -
    1  &&  slope  <=  0)  ||  (inputX  <  0  &&  inputY  <=  0  &&  slope
    >=  0  &&  slope  <  1))                 {
75.                 if  (PlayerPrefs.GetInt("soundEffect")  ==  1)
                    bgMusic.GetComponents < AudioSource > ()[1].enabled  =
     true;
76.                 StartCoroutine(move_left());
77.                 stateGuide  =  0;
78.             }
79.             else  isMove  =  false;
80.         }
81.         else  if  (stateGuide  ==  2)             {
82.             if  ((inputX  >=  0  &&  inputY  <  0  &&  slope  <  -
    1)  ||  (inputX  <=  0  &&  inputY  <  0  &&  slope  >  1))
         {
83.                 if  (PlayerPrefs.GetInt("soundEffect")  ==  1)
        bgMusic.GetComponents < AudioSource > ()[1].enabled  =  true;

84.                 StartCoroutine(move_down());
85.                 stateGuide  =  0;
86.             }
87.             else  isMove  =  false;
88.         }
89.         else  isMove  =  false;
90.     }
91. }
```

**Kode Sumber 4.1 Implementasi pergerakan karakter**

### 4.2.2. Implementasi Berpindah Waktu

Pemain dapat melakukan perpindahan *maze* ke masa lalu ataupun sebaliknya dengan cara menyentuh tombol perpindahan waktu. Saat pemain menyentuhnya maka pemain berpindah waktu dan tampilan akan berpindah menandakan perpindahan waktu berhasil. Pemain dapat berpindah waktu tanpa adanya batasan jumlah perpindahan waktu. Namun, pemain tidak dapat melakukan perpindahan waktu jika berada di tempat bibit ditanam.

Implementasi perpindahan waktu sebagaimana yang telah dijelaskan dapat dilihat pada Kode Sumber 4.2.

```
1.  void teleport() {
2.      if (PlayerPrefs.GetInt("soundEffect") == 1)         AudioSource
    .PlayClipAtPoint(Time1, Camera.main.transform.position);
3.      map.isFuture = !map.isFuture;
4.      mapPast.SetActive(!mapPast.activeInHierarchy);
5.      mapFuture.SetActive(!mapFuture.activeInHierarchy);
6.      ed.countTeleport--;
7.  }
```

**Kode Sumber 4.2 Perpindahan waktu**

### 4.2.3. Implementasi Mendapatkan Poin

Pemain dapat memenangkan permainan saat berhasil mendapatkan seluruh poin harapan yang ada. Pemain bisa mendapatkan poin harapan dengan cara karakter pemain bersentuhan dengan poin harapan. Implementasi kondisi menang permainan sebagaimana yang telah dijelaskan dapat dilihat pada Kode Sumber 4.3.

```
1.  public class PointHope : MonoBehaviour {
2.      private EndGame ed;
3.      private MoveChar itsa;
4.      MappingGround map;
5.      public AudioClip sound;
6.      void Start () {
7.          ed = GameObject.FindGameObjectWithTag("End").GetComponent < E
    ndGame > ();
8.          itsa = GameObject.FindGameObjectWithTag("Itsa").GetComponent
    < MoveChar > ();
9.          map = GameObject.Find("MapGround").GetComponent < MappingGrou
    nd > ();
10.     }
11.     void OnTriggerEnter2D(Collider2D coll)     {
12.         if (coll.tag == "Itsa")         {
13.             if (PlayerPrefs.GetInt("soundEffect") == 1)
        AudioSource.PlayClipAtPoint(sound, Camera.main.transform.position)
    ;
14.             map.stageFuture[itsa.positionItsaX, itsa.positionItsaY] =
      0;
```

```
            ed.countPoin--;
            if (ed.countPoin == 0)            {
                itsa.isMove = false;
                itsa.ani.Play("ItsaSenang");
                ed.isEnd = true;
                ed.End();
            }
            Destroy(gameObject);
        }
    }
}
```

**Kode Sumber 4.3 Mendapatkan poin**

### 4.2.4. Implementasi Menanam Bibit

Pemain dapat menanam bibit saat berada di *maze* masa lalu. Pemain dapat menanam di area yang disediakan yaitu tempat penanaman bibit. Pemain dapat menanam bibit dengan cara karakter pemain melewati atau berhenti di tempat penanaman bibit. Implementasi dapat dilihat pada Kode Sumber 4.4.

```
void Planting()    {
    mg.stageFuture[positionItsaX, positionItsaY] = 4;
    mg.stagePast[positionItsaX, positionItsaY] = 13;
    mg.deleteInst();
    mg.MapRealization();
}
```

**Kode Sumber 4.4 Penanaman bibit**

### 4.2.5. Implementasi *Stage*

Pengaturan stage yang mencakup segala hal yang berhubungan dengan pengaturan *load stage* dalam permainan. Kode dapat dilihat di Kode Sumber 4.5.

```
public void GoToStage(int stage)    {
    SceneManager.LoadScene(stage);
}
public void OpenWindow(GameObject TempWindow)    {
    TempWindow.SetActive(true);
}
public void CloseWindow(GameObject TempWindow)    {
    TempWindow.SetActive(false);
```

```
9. }
```

**Kode Sumber 4.5 Pemilihan stage**

## 4.3. Implementasi Antarmuka

### 4.3.1. Implementasi *Main Menu*

Implementasi dari Main Menu mencakup tombol menuju ke *stage selection* yaitu dengan menggunakan tombol ‘play’ dan tombol untuk keluar aplikasi, serta tombol untuk pengaturan suara. *Screenshot* dari *Main Menu* dapat dilihat pada Gambar 4.1.


**Gambar 4.1 Implementasi tampilan *Main Menu***

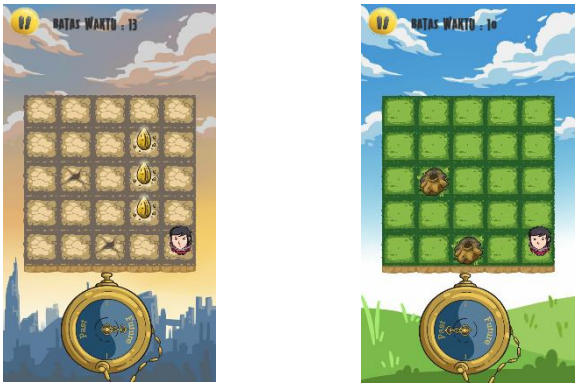
### 4.3.2. Implementasi Stage Selection

Implementasi *Stage Selection* mencakup daftar *substage* yang dapat dimainkan dan dapat diakses setelah pemain memilih stage, serta pilihan untuk kembali ke main menu. *Screenshot* dari Level Menu dapat dilihat pada Gambar 4.2.


**Gambar 4.2 Implementasi Tampilan *Stage Selection***

### 4.3.3. Implementasi Gameplay Stage

Implementasi *Gameplay Stage* mencakup komponen stage berupa karakter pemain, tombol perpindahan waktu, *obstacle*, tempat penanaman bibit, poin dan susunan *puzzle* yang direalisasikan dari array yang terbentuk oleh algoritma Hunt-and-Kill. Pada bagian atas layar permainan terdapat batas waktu untuk menyelesaikan stage, selain itu juga terdapat tombol pause yang dapat mengarahkan pemain kembali *stage selection*. *Screenshot* dari *gameplay* dapat dilihat pada Gambar 4.3.




**Gambar 4.3 Implementasi tampilan *Gameplay* Stage**

## 4.4. Implementasi Hunt-and-Kill

Pada bagian ini menjelaskan bagaimana menerapkan aturan algoritma Hunt-and-Kill untuk digunakan ke aplikasi permainan dalam pembentukan array susunan *puzzle* stage. Penerapan algoritma Hunt-and-Kill akan berjalan setelah konfigurasi StageController dilakukan. Code dari stage controller dapat dilihat pada Kode Sumber 4.6.

```
1.  public  class  StageController :  MonoBehaviour  {
2.      public  MapGenerator  stageGenerator;
3.      public  int[, ]  stageFuture;
4.      public  int[, ]  stagePast;
5.      int  rowLength  =  0;
6.      int  columnLength  =  0;
7.      public  int  leRow;
8.      public  int  leColumn;
9.      public  int  winhope;
10.     public  GameObject  pastMap;
11.     public  GameObject  Future;
12.     void  Start ()  {
13.         Future  =  GameObject.Find("Future");
14.         GeneratePuzzle();
15.         GeneratePuzzlePast();
16.     }
```

```
17.     void GeneratePuzzle()     {
18.         int temp = 0;
19.         rowLength = leRow;
20.         columnLength = leColumn;
21.         stageFuture = new int[rowLength, columnLength];
22.         stageGenerator.Generate(leRow, leColumn);
23.         for (int i = 0; i < rowLength; i++)         {
24.             for (int j = 0; j < columnLength; j++)             {
25.                 temp = stageGenerator.getPuzzle(i, j);
26.                 if (temp == 6)                     temp = -
    1;
27.                 else if (temp == 3)                     temp = 1;
28.                 else if (temp == 1)                     temp = 0;
29.                 else if (temp == 4)                     temp = 3;
30.                 else if (temp == 2)                     temp = 6;
31.                 stageFuture[i, j] = temp;
32.             }
33.         }
34.     }
35.     void GeneratePuzzlePast()     {
36.         int temp = 0;
37.         stagePast = new int[rowLength, columnLength];
38.         for (int i = 0; i < rowLength; i++)         {
39.             for (int j = 0; j < columnLength; j++)
    {
40.                 temp = stageGenerator.getPuzzlePast(i, j);
41.                 if (temp == 1)                     temp = 7;
42.                 else if (temp == 4)                     temp = 2;
43.                 else if (temp == 6)                     temp = 7;
44.                 else if (temp == 2)                     temp = 5;
```

```
45.                 stagePast[i,  j]  =  temp;
46.             }
47.         }
48.     }
49. }
```

**Kode Sumber 4.6 StageController.cs**

StageController akan melakukan konfigurasi terhadap variabel-variabel yang dimasukkan untuk tiap stagenya. Contoh konfigurasi dapat dilihat pada Gambar 4.4.


**Gambar 4.4 Penerapan konfigurasi stage**

Pada konfigurasi StageController Le Row dan Le Column merupakan dimensi dari *puzze* dan Winhope merupakan jumlah dari poin yang akan dimunculkan dalam *puzzle* sesuai yang tertera di bagian perancangan stage pada bagian 3.2.1. Sedangkan Stage Generator akan membuat realisasi dari array yang didapatkan dari hasil algoritma Hunt-and-Kill.

Algoritma Hunt-and-Kill diterapkan dalam bentuk sebuah class bernama MapGenerator.cs. Tahap ‘Hunt’ dan ‘Kill’ masuk kedalamnya dan berjalan sesuai dengan perancangan yang di jelaskan dalam bagian perancangan 3.2.3.Implementasi Hunt-and-Kill dapat di lihat pada Kode Sumber 1.A pada bagian Lampiran A Tugas Akhir ini.

Kode sumber yang menerapkan algoritma Hunt-and-Kill akan menghasilkan array yang kemudian akan direalisasikan menjadi tampilan permainan. Penjelasan mengenai fungsi yang

terdapat pada Kode Sumber 1.A yang ada pada lampiran terdapat pada tabel 4.1.

**Tabel 4.1 Penjelasan metode MapGenerator**

| No | Nama Fungsi | Penjelasan Fungsi |
|---|---|---|
| **1** | HistReset() | Reset seluruh array *history* penelusuran |
| **2** | Generate() | Fungsi utama untuk mengatur jalannya class |
| **3** | SeedPlace() | Mengubah sejumlah *obstacle* menjadi tempat penanaman bibit |
| **4** | ClearHope() | Mengubah array poin menjadi ground |
| **5** | ResetPath() | Mengubah seluruh nilai array kembali menjadi 0 |
| **6** | SetFutureMap() | Set *puzzle* masa depan |
| **7** | get*Puzzle*() | Melakukan *return* isi array susunan *puzzle* masa depan |
| **8** | get*Puzzle*Past() | Melakukan *return* isi array susunan *puzzle* masa lalu |
| **9** | ResetMap() | Melakukan reset pada array *puzzle* masa lalu dan masa depan |
| **10** | PlaceHope() | Meletakkan poin pada array |
| **11** | Generate*Puzzle*() | Inisialisasi Algoritma Hunt-and-Kill |
| **12** | StageNormalization() | Mengubah titik potensial pemberhentian karakter menjadi ground |
| **13** | CheckRightLeft() | Cek kanan dan kiri dari titik penelusuran |
| **14** | CheckTopBottom() | Cek atas dan bawah dari titik penelusuran |
| **15** | HuntPoint() | Melakukan ‘Hunt’ |
| **16** | FirstHunt() | Melakukan ‘First Hunt’ |
| **17** | DoHunt() | Inisialisasi ‘Hunt’ |
| **18** | DoKill() | Inisialisasi ‘Kill’ |
| **19** | CheckTurn() | Cek apakah penelusuran akan menyebabkan belokan |
| **20** | *Obstacle*Mark() | Meletakkan *Obstacle* |
| **21** | Movekill() | Melakukan ‘Kill’ |
| **22** | WayObserve() | Melakukan pengecekan arah yang dapat ditelusuri |

## 4.5. Implementasi Realisasi *puzzle*

Algoritma Hunt-and-Kill menghasilkan array yang akan digunakan untuk merealisasikan tampilan *puzzle* stage. Array yang dihasilkan akan digunakan untuk susunan *puzzle* masa depan dan *puzzle* masa lalu dengan penerapan yang berbeda. Perbedaan penerapannya terletak pada tempat penanaman bibi dan poin. Tempat penanaman bibit hanya ada di *puzzle* masa lalu, sedangkan di *puzzle* masa depan hanya berupa jalan biasa. Poin hanya ada di *puzzle* masa depan sedangkan di *puzzle* masa lalu poin tidak

ditampilkan. Titik atau posisi yang tidak di cek akan di realisasikan sebagai tampilan *ground* atau jalan. Method yang menampilkan realisasi *puzzle* dapat dilihat pada Kode Sumber 4.7.

```
1. void mapFuture(int i, int j, float xPoint, float yPoint, float
   width)    {
2.     if (stageFuture[i, j] != -
   1)            obj = Instantiate(ground[stageFuture[i, j]], new Ve
   ctor3(0, 0), transform.rotation);
3.     else if (stageFuture[i, j] == -1)        {
4.         obj = Instantiate(ground[0], new Vector3(0, 0), transform
   .rotation);
5.         stageFuture[i, j] = 0;
6.         if (PlayerPrefs.GetInt("karakter") == 0) {
7.             GameObject charItsa = Instantiate(Itsa, new Vector3(0,
    0), transform.rotation);
8.             charItsa.SetActive(true);
9.             charItsa.transform.SetParent(gameObject.transform);

10.            charItsa.GetComponent < RectTransform > ().sizeDelta = ne
   w Vector2(width - 20, width - 10);
11.            charItsa.transform.localPosition = new Vector3(xPoint +
    width / 15, yPoint + (width / 4), 0);
12.            charItsa.GetComponent < MoveChar > ().positionItsaX = i;

13.            charItsa.GetComponent < MoveChar > ().positionItsaY = j;

14.        }
15.    }
16.    obj.transform.SetParent(gameObject.transform.GetChild(0));

17.    obj.GetComponent < RectTransform > ().sizeDelta = new Vector2(wi
   dth, width);
18.    obj.transform.localPosition = new Vector3(xPoint, yPoint, 0);

19. }
20. void mapPast(int i, int j, float xPoint, float yPoint, float
   width)    {
21.    if (stagePast[i, j] == 0)            obj = Instantiate(groun
   d[7], new Vector3(0, 0), transform.rotation);
22.    else            obj = Instantiate(ground[stagePast[i, j]], new
    Vector3(0, 0), transform.rotation);
```

```
23.     obj.transform.SetParent(gameObject.transform.GetChild(1));

24.     obj.GetComponent < RectTransform > ().sizeDelta  =  new  Vector2(wi
    dth,  width);
25.     obj.transform.localPosition  =  new  Vector3(xPoint,  yPoint,  0);

26. }
```

**Kode Sumber 4.7 Implementasi realisasi *puzzle***

# BAB V
# PENGUJIAN DAN EVALUASI

## 5.1. Lingkungan Pengujian

Lingkungan pelaksanaan uji coba meliputi perangkat keras dan perangkat lunak yang akan digunakan pada sistem ini. Spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam rangka uji coba perangkat lunak ini dicantumkan pada Tabel 5.1.

**Tabel 5.1 Lingkungan pengujian**

| | |
|---|---|
| Perangkat Keras | Samsung Galaxy S7<br>CPU: Octa-core 2.6GHz<br>Memori : 4.00 GB |
| Perangkat Lunak | Sistem Operasi Android 7.0 |

## 5.2. Pengujian Fungsionalitas

Untuk mengetahui kesesuaian keluaran dari tiap tahap dan langkah penggunaan fitur terhadap skenario yang dipersiapkan, maka dibutuhkan pengujian fungsionalitas.

Pengujian ini dilakukan untuk menguji apakah fungsionalitas yang diidentifikasi benar-benar diimplementasikan dan bekerja sebagaimana seharusnya. Pengujian juga dilakukan untuk mengetahui kesesuaian setiap tahapan atau langkah penggunaan fitur terhadap skenario yang dipersiapkan. Pengujian dilakukan dengan metode *black-box*.

### 5.2.1. Skenario Pengujian

Skenario pengujian fungsionalitas digunakan untuk memberikan tahap-tahap dalam pengujian sistem. Skenario tertera pada Tabel 5.2.

**Tabel 5.2 Pengujian aplikasi permainan**

| | |
|---|---|
| Kondisi Awal | Pengguna berada pada layar Main Menu |

| | |
|---|---|
| Prosedur Pengujian | Pengguna masuk ke *Stage Selection*, memilih Stage yang akan dimainkan, lalu memainkan permainan hingga selesai |
| Hasil yang diharapkan | Pengguna berhasil menyelesaikan permainan dan fungsionalitas permainan berjalan dengan lancar. |
| Hasil yang diperoleh | Pengguna berhasil menyelesaikan permainan dan fungsionalitas berjalan lancar. |
| Kesimpulan | Pengujian berhasil |

### 5.2.1.1. Pengujian Main Menu dan Stage Selection

Pengujian dimulai ketika pengguna telah masuk ke layar Main menu seperti yang terlihat pada Gambar 5.1. Pemain memilih menekan tombol 'Play' untuk masuk ke dalam layar *Stage Selection*.


**Gambar 5.1 Tampilan *Main Menu***

Setelah menekan tombol ‘Play’ maka sistem menampilkan layar *Stage Selection* seperti yang terlihat pada Gambar 5.2.


**Gambar 5.2 Tampilan *Stage Selection***

Setelah masuk ke tampilan *stage selection*, pemain dapat memilih stage yang ingin dimainkan dengan memilih salah satu stage. Saat pemain memilih stage maka akan muncul substage yang akan dipilih pemain, pemain dapat memilih substage yang ingin dimainkan. Selain itu di tampilan stage selection juga terdapat tombol ‘back’ yang dapat digunakan untuk kembali ke tampilan *Main Menu.*

Setelah melakukan pengujian, sistem berhasil masuk ke Game Menu dan melakukan load sesuai stage yang dipilih. Selain itu sistem juga berhasil kembali ke *Main Menu* ketika pengguna menekan tombol ‘back’. Sehingga dapat disimpulkan bahwa pengujian untuk layar *Main Menu* dan *Stage Selection* berhasil.

### 5.2.1.2. Pengujian Pergerakan Karakter Pemain

Pengujian dimulai ketika pemain telah masuk ke *gameplay stage*. Pemain dapat menggerakkan karakter dengan cara melakukan *swipe* layar permainan dan terus bergerak hingga menemui batas *puzzle* atau bertemu dengan *obstacle*. Setelah pengujian, karakter pemain dapat bergerak dengan baik sesuai *gameplay* permainan. sehingga dapat disimpulkan bahwa pengujian untuk pergerakan karakter pemain berhasil. Berikut hasil pengujian ini dapat dilihat pada Gambar 5.3.


**Gambar 5.3 Tampilan pergerakakan karakter**

### 5.2.1.3. Pengujian Perpindahan Waktu

Pengujian dimulai ketika pemain telah masuk ke *gameplay stage*. Pemain dapat melakukan perpindahan waktu ke masa lalu ataupun sebaliknya dan mendapatkan tampilan yang berbeda dengan cara menyentuh tombol perpindahan waktu. Setelah diuji disimpulkan bahwa perpindahan waktu berhasil. Berikut hasil dari pengujian ini dapat dilihat pada Gambar 5.4.


**Gambar 5.4 Tampilan perpindahan waktu maze ‘masa depan’ (kiri) dan maze ‘masa lalu’(kanan)**

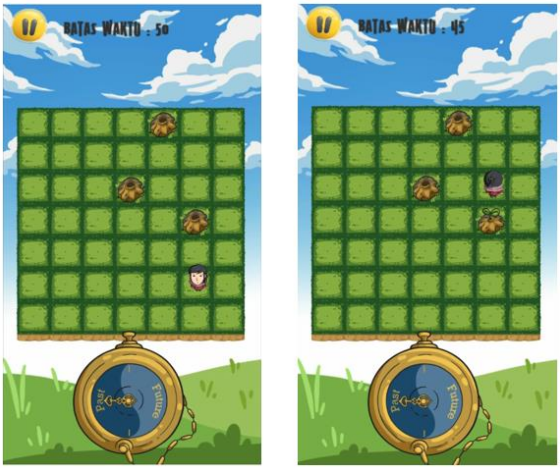
### 5.2.1.4. Pengujian Mendapatkan Poin

Pengujian dimulai ketika pemain telah masuk ke *gameplay stage*. Untuk kondisi awalnya poin harapan akan ada pada stage di posisi tertentu. Karakter pemain melewati poin harapan dan poin harapan akan hilang menandakan poin harapan telah berhasil didapatkan oleh pemain. Berikut hasil dari pengujian ini dapat dilihat pada gambar 5.5.


**Gambar 5.5 Pemain mendapatkan poin saat bergerak**

### 5.2.1.5. Pengujian Penanaman Bibit

Pengujian dimulai ketika pemain telah masuk ke *gameplay stage*, dan telah berpindah waktu menjadi ‘masa lalu’. Untuk kondisi awalnya tempat penanaman bibit terletak di posisi tertentu pada stage. Karakter pemain melewati atau berhenti di tempat penanaman bibit dan muncul tampilan tempat penanaman bibit berubah menandakan bibit telah berhasil ditanam oleh pemain. Berikut hasil dari pengujian ini dapat dilihat pada gambar 5.6.




**Gambar 5.6 Bibit belum ditanam(kiri) dan setelah ditanam(kanan)**

Setelah penanaman bibit dilkukan maka di ‘masa depan’ muncul objek pohon. Objek pohon yang muncul berada di posisi yang sama dengan tempat pemain menanam bibit di ‘masa lalu’. Setelah diuji disimpulkan bahwa pengujian penananman bibit berhasil dan pohon sebagai *obstacle* berhasil muncul di *puzzle* masa depan. Berikut hasil dari pengujian ini dapat dilihat pada gambar 5.7.


**Gambar 5.7 Pohon muncul di tampilan masa depan**

### 5.2.1.6. Pengujian Menang dan Kalah

Pengujian ini dimulai dengan menyelesaikan stage permainan. Saat permain telah mendapatkan seluruh poin yang ada dalam stage dan memenangkan permainan maka *window* atau tampilan menang akan muncul, menandakan pemain telah sukses menyelesaikan stage. Tampilan pemain menang dan tampilan stage menang muncul dapat dilihat pada Gambar 5.8.


**Gambar 5.8 Tampilan saat pemain menang**

Pemain akan dinyatakan kalah saat waktu habis. Saat waktu habis maka akan muncul *window* atau tampilan kalah atau

stage gagal. Tampilan pengujian pemain kalah karena batas waktu telah habis dan pemain tidak berhasil mendapatkan poin sehinggan akan muncul tampilan stage gagal dapat dilihat pada Gambar 5.9.


**Gambar 5.9 Batas waktu habis dan pemain kalah**

### 5.2.2. Hasil Pengujian

Hasil uji fungsionalitas yang telah dilakukan berdasarkan skenario sebelumnya, menunjukkan bahwa semua fungsionalitas permainan berjalan dengan baik dan sesuai dengan sebagaimana mestinya yang telah dibuat pada tahap perancangan. Hasil uji fungsionalitas dapat dilihat pada Tabel 5.3.

**Tabel 5.3 Hasil pengujian fungsionalitas**

| No | Pengujian | Hasil Pengujian |
|---|---|---|
| 1 | *Main Menu* dan *Stage Selection* | Berhasil |
| 2 | Pergerakan karakter pemain | Berhasil |
| 3 | Perpindahan waktu | Berhasil |
| 4 | Mendapatkan poin | Berhasil |
| 5 | Penanaman bibit | Berhasil |
| 6 | Menang dan Kalah | Berhasil |

## 5.3. Pengujian Penerapan Algoritma Hunt-and-Kill

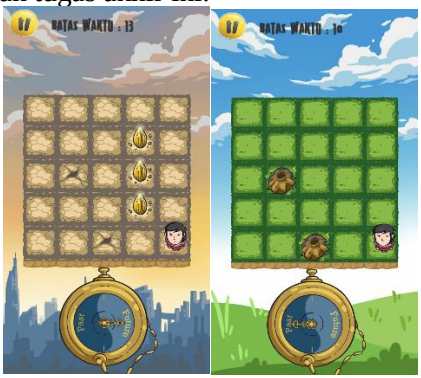
Pengujian penerapan algoritma Hunt-and-Kill pada game Plant the Future mencakup seberapa efektif algoritma tersebut digunakan sebagai penyusun *puzzle* yang dapat diselesaikan. Pengujian ini dilakukan oleh satu orang pengguna.

### 5.3.1. Skenario Pengujian

Pengujian dilakukan pada 30 stage pada game 'Plant the Future' yang menerapkan algoritma Hunt-and-Kill. Pengujian dilakukan dengan memainkan setiap stage sebanyak lima kali secara beruntun dan diamati apakah stage yang dibuat dapat diselesaikan atau tidak. Penulis secara manual menguji stage yang merupakan *puzzle* hasil dari penerapan algoritma Hunt-and-Kill.

### 5.3.2. Hasil Pengujian

Hasil yang diuji merupakan tampilan dari realisasi array menjadi tampilan permainan. Salah Satu contoh tampilan, hasil dari penggunaan algoritma Hunt-and-Kill dapat dilihat pada gambar 5.10. Seluruh tampilan dari stage yang diuji dapat dilihat pada bagian lampiran tugas akhir ini.




**Gambar 5.10 Tampilan stage pengujian**

Pengujian dilakukan pada seluruh stage yang menerapkan algoritma Hunt-and-Kill yaitu berjumlah 30 stage dengan dimensi puzzle yang berbeda-beda. Hasil pengujian 30 stage sebanyak lima

kali masing-masingnya dengan total 150 stage uji coba, menghasilkan hasil seperti ditunjukkan pada Table 5.4.

**Tabel 5.4 Hasil pengujian**

| No | Stage | Jumlah Pengujian Stage | Jumlah berhasil diselesaikan |
|---|---|---|---|
| 1 | 2 | 5 | 5 |
| 2 | 3-1 | 5 | 5 |
| 3 | 3-2 | 5 | 5 |
| 4 | 4-1 | 5 | 5 |
| 5 | 4-2 | 5 | 5 |
| 6 | 4-3 | 5 | 5 |
| 7 | 5-1 | 5 | 5 |
| 8 | 5-2 | 5 | 5 |
| 9 | 5-3 | 5 | 5 |
| 10 | 5-4 | 5 | 5 |
| 11 | 6-1 | 5 | 5 |
| 12 | 6-2 | 5 | 5 |
| 13 | 6-3 | 5 | 5 |
| 14 | 6-4 | 5 | 5 |
| 15 | 6-5 | 5 | 5 |
| 16 | 7-1 | 5 | 5 |
| 17 | 7-2 | 5 | 5 |
| 18 | 7-3 | 5 | 5 |
| 19 | 7-4 | 5 | 5 |
| 20 | 7-5 | 5 | 5 |
| 21 | 8-1 | 5 | 5 |
| 22 | 8-2 | 5 | 5 |
| 23 | 8-3 | 5 | 5 |
| 24 | 8-4 | 5 | 5 |
| 25 | 8-5 | 5 | 5 |
| 26 | 9-1 | 5 | 5 |
| 27 | 9-2 | 5 | 5 |
| 28 | 9-3 | 5 | 5 |
| 29 | 9-4 | 5 | 5 |
| 30 | 9-5 | 5 | 5 |

Dari hasil pengujian didapatkan bahwa dari 150 stage yang diuji seluruhnya berhasil diselesaikan. Sehingga Hunt-and-Kill

dapat dikatakan berhasil diterapkan pada permainan ‘Plant the Future’ sebagai algoritma penyusun stage.

## 5.4. Pengujian Pengguna

Pengujian pengguna adalah untuk mendapatkan informasi mengenai pendapat pengguna mengenai game ‘Plant the Future’ dan stage dinamis yang diterapkan pada game ‘Plant the Future’. Pengujian dilakukan ke tiga pengguna.

### 5.4.1. Skenario Pengujian

Pengujian dilakukan oleh pengguna (pemain) dengan memainkan game ‘Plant the Future’. Pemain memainkan satu stage yang sama berulang-ulang. Kemudian pemain diberikan *form* berisi beberapa pertanyaan. Pertanyaan yang diberikan meliputi pendapatnya mengenai game ‘Plant the Future’, pendapat pengguna mengenai stage dinamis yang diterapkan, dan masukan untuk pengembangan aplikasi permainan lebih lanjut.

### 5.4.2. Hasil Pengujian

Setelah dilakukan pengujian oleh pengguna, didapatkan beberapa informasi. Pengguna berpendapat bahwa dari segi penggunaan aplikasi permainan, ‘Plant the Future’ sudah cukup baik. Aplikasi permainan sudah cukup mudah untuk dimengerti dan sudah nyaman untuk digunakan. Gameplay permainan juga menarik dan permainan sudah dapat berjalan sebagaimana mestinya. Informasi stage yang berulang kali dicoba pemain ditunjukkan pada Tabel 5.5.

**Tabel 5.5 Hasil Pengujian Pengguna**

| Pengguna | Stage yang dicoba | Jumlah Dimainkan | Jumlah Menang |
|---|---|---|---|
| **1** | 4-1 | 10 | 7 |
| **2** | 4-3 | 7 | 6 |
| **3** | 3-1 | 4 | 4 |

Selama bermain di stage yang sama, permainan berhasil menghasilkan stage yang berbeda setiap kali pengguna memainkannya. Adanya batas waktu membuat pemain lebih

tertantang untuk menyelesaikan permainan. Pengguna juga berpendapat bahwa dengan stage dinamis yang menghasilkan susunan puzzle membuat permainan lebih tidak membosankan jika dimainkan berulang kali.

# BAB VI
# KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

## 6.1. Kesimpulan

Dari proses penerapan algoritma Hunt-and-Kill untuk perancangan *puzzle* pada game Plant the Future dapat diambil kesimpulan sebagai berikut:

1. 'Plant the Future' berhasil dirancang dengan baik menerapkan pergerakan karakter permainan yang diadopsi dari permainan yang mirip dan menggunakan *gameplay* aturan permainan yang dirancang sendiri oleh penulis.
2. Aturan perancangan puzzle menggunakan algoritma Hunt-and-Kill diterapkan dengan memodifikasi algoritma yaitu menambahkan beberapa kondisi untuk menyesuaikan Hunt-and-Kill dan *gameplay* game 'Plant the Future'.
3. Penerapan algoritma Hunt-and-Kill dalam perancangan puzzle dalam game diterapkan dalam *game engine*, dan permainan dapat berjalan sebagaimana mestinya dalam perangkat mobile.

## 6.2. Saran

Saran yang diberikan terkait pengembangan pada Tugas Akhir ini berdasarkan perancangan, implementasi, dan uji coba adalah:

1. Menambah parameter tingkat kesulitan dalam penyusunan *puzzle* stage.
2. Memperbanyak jumlah pengujian terhadap setiap stage.
3. Untuk pengembangan lebih lanjut disarankan agar menggabungkan penggunaan algoritma Hunt-and-Kill

dengan algoritma lain untuk menciptakan tingkat kesulitan dengan parameter yang lebih banyak dan terarah, bukan hanya dari sisi dimensi stage.

4. Menambah fitur mode permainan dan power up.
5. Petunjuk permainan lebih diperjelas.

# DAFTAR PUSTAKA


[1] Ijsselsteijn, Wijanand et al. (2008). Measuring the Experience of Digital Game Enjoyment. Proceedings of Measuring Behavior 2008, 6th International Conference on Methods and Techniques in Behavioral Research. Maastricht, Netherlands, August 26-29, 2008.

[2] Dix, Alan et al. (2004). Human-Computer Interaction 3rd. England: Pearson Education Limited.

[3] Unity. "CREATE GAMES, CONNECT WITH YOUR AUDIENCE, AND ACHIEVE SUCCESS". Unity Technologies, 2017. [Online]. Available: http://unity3d.com/unity. [Diakses 2 September 2017].

[4] Kemppainen, Jaakko. (2014). Designing a Knowledge Based Puzzle Game. Finlandia: Aalto University.

[5] Buck, Jamis. (2015). *Maze* for Programmers. United States of America: The Pragmatic Programmers.

*[Halaman ini sengaja dikosongkan]*

# BIODATA PENULIS

Penulis lahir di Pekanbaru, 25 September 1996, merupakan anak pertama dari tiga bersaudara. Dalam menjalani pendidikan semasa hidup, penulis menempuh pendidikan di TK An-Nur Pekanbaru, SDN 001 Sail Pekanbaru, SMPN 4 Pekanbaru, SMAN 8 Pekanbaru dan S1 Departemen Informatika Institut Teknologi Sepuluh Nopember (ITS) pada rumpun Interaksi Grafika dan Seni (IGS).

Selama menjadi mahasiswa, penulis ikut dalam Himpunan Mahasiswa Teknik Komputer Informatika ITS, dalam departemen Ristek pada tahun kedua, dilanjutkan dengan tahun ketiga di departemen Teknologi sebagai staf ahli, staf ahli departemen Syiar KMI(Keluarga Muslim Informatika) dan staf Schematics 2016 di biro NPC. Penulis telah menghasilkan beberapa karya aplikasi permainan diantaranya: 'Logic Bullet','Logic Bullet Extented', 'Get the Trash', 'EnCollect' dan 'Plant the Future'. Berkat kegemarannya di ranah game development, penulis berkesempatan mengikuti beberapa kompetisi bersama tim fragments. Penulis mengikuti beberapa kompetisi seperti ajang Gemastik 9 yang diselenggarakan di Universitas Indonesia berhasil menjadi finalis dengan membawakan game 'Logic Bullet', mendapatkan juara 1(satu) pada ajang Game Development MAGE 2017 dengan membawakan game 'Logic Bullet Extended', mendapatkan juara 1(satu) pada ajang Game Development FTIF Festival 2017 dengan membawakan game 'Get the Trash' dan mendapatkan medali emas kategori aplikasi permainan pada ajang Gemastik 10 di Universitas Indonesia dengan membawakan game 'Plant the Future'.

*[Halaman ini sengaja dikosongkan]*

# LAMPIRAN A

```
1.  public  class  MapGenerator :  MonoBehaviour  {
2.      public  int  sumHope;
3.      int  chunt = 1;
4.      bool  future  =  true;
5.      bool  PlayerWay  =  true;
6.      int  row  =  5;
7.      int  column  =  5;
8.      int  rowPoint;
9.      int  columnPoint;
10.     int  count;
11.     bool  character = true;
12.     int  crow;
13.     int  ccol;
14.     int  obstacle  =  0;
15.     int  test  =  20;
16.     int[, ]  map;
17.     int[, ]  pastMap;
18.     int[, ]  futureMap;
19.     int[, ]  dummyMap;
20.     int[]  randomDirection;
21.     obstacles[]  obs;
22.     int[]  histRow;
23.     int[]  histCol;
24.     int  hisDir  =  0;
25.     int  frow;
26.     int  fcolumn;
27.     int  through;
28.     int  nowTurn;
29.     int  direction  =  0;
30.     /* atas = 1     * kanan = 2     * kiri = 3     * bawah = 4     */
31.
32.     bool  isHunt  =  false;
33.     bool  isKill  =  false;
34.     bool  first  =  true;
35.
36.     struct  obstacles     {
37.         public  int  x;
38.         public  int  y;
39.     }
40.     void  HistReset()     {
```

```
41.         hisDir  =  0;
42.         histRow[0]  =  -1;
43.         histCol[0]  =  -1;
44.         histRow[1]  =  -1;
45.         histCol[1]  =  -1;
46.         direction  =  0;
47.     }
48.     public  void  Generate(int  newRow,  int  newCol)     {
49.         row  =  newRow;
50.         column  =  newCol;
51.         obs  =  new  obstacles[65];
52.         direction  =  0;
53.         histRow  =  new  int[2];
54.         histCol  =  new  int[2];
55.         dummyMap  =  new  int[row,  column];
56.         Debug.Log("Now Generating Maze!");
57.         map  =  new  int[row,  column];
58.         ResetMap();
59.         GeneratePuzzle();
60.         SetFutureMap();
61.         ResetPath();
62.         future  =  false;
63.         PlayerWay  =  true;
64.         Debug.Log("END");
65.         ClearHope();
66.         SeedPlace();
67.     }
68.     void  SeedPlace()     {
69.         int  tree  =  (obstacle  *  75)  /  100;
70.         int  seedy  =  0;
71.         if (obstacle < 3)         {
72.             for (int  i = 0; i < obstacle; i++)             {
73.                 futureMap[obs[i].x,  obs[i].y]  =  4;
74.                 map[obs[i].x,  obs[i].y]  =  4;
75.             }
76.         }
77.         else  if (obstacle > 2)         {
78.             while (tree > 0)             {
79.                 seedy  =  Random.Range(0,  obstacle  -  1);
```

```
80.                 if (obs[seedy].x != -
    1)                 {
81.                     map[obs[seedy].x,  obs[seedy].y]  =  4;
82.                     futureMap[obs[seedy].x,  obs[seedy].y]  =  4;
83.                     obs[seedy].x  =  -1;
84.                     tree--;
85.                 }
86.             }
87.         }
88.     }
89.     void  ClearHope()     {
90.         for (int  i = 0; i < row; i++)         {
91.             for (int  j = 0; j < column; j++)             {
92.                 if (map[i, j] == 3)                 {
93.                     map[i,  j]  =  1;
94.                 }
95.             }
96.         }
97.     }
98.     void  ResetPath()     {
99.         for (int  i = 0; i < row; i++)         {
100.                for (int  j = 0; j < column; j++)             {
101.                    if ( map[i, j]  ==  1)                 {
102.                        map[i,  j]  =  0;
103.                    }
104.                }
105.            }
106.        }
107.        void  SetFutureMap()     {
108.            futureMap  =  new  int[row,  column];
109.            for (int  i  =  0 ;  i  <  row ;  i++)         {
110.                for (int  j = 0; j < column; j++)             {
111.                    futureMap[i,  j]  =  map[i,  j];
112.                }
113.            }
```

```
114.        }
115.        public int getPuzzle(int x, int y)    {
116.            return futureMap[x, y];
117.        }
118.        public int getPuzzlePast(int x, int y)    {
119.            return map[x, y];
120.        }
121.        void ResetMap()    {
122.            for (int p = 0; p < row; p++)        {
123.                for (int q = 0; q < column; q++)            {

124.                    map[p, q] = 0;
125.                }
126.            }
127.            for (int p = 0; p < row; p++)        {

128.                for (int q = 0; q < column; q++)            {

129.                    dummyMap[p, q] = 0;
130.                }
131.            }
132.        }
133.        void PlaceHope()    {
134.            while (sumHope > 0)        {
135.                int rnew = Random.Range(0, row - 1);

136.                int cnew = Random.Range(0, column - 1);

137.                if (map[rnew, cnew] == 1)            {

138.                    map[rnew, cnew] = 3;
139.                    dummyMap[rnew, cnew] = 3;
140.                    sumHope--;
141.                }
142.            }
143.        }
144.        void GeneratePast()    {
145.            pastMap = new int[row, column];
146.        }
147.        void GeneratePuzzle()    {
148.            isHunt = true;
149.            while (isHunt)        {
150.                HistReset();
```

```
151.                direction = 0;
152.                DoHunt();
153.            }
154.            StageNormalization();
155.            PlaceHope();
156.        }
157.        void StageNormalization()    {
158.            for (int p = 0; p < row; p++)        {
159.                for (int q = 0; q < column; q++)            {
160.                    if (map[p, q] == 5)                {
161.                        map[p, q] = 1;
162.                    }
163.                }
164.            }
165.        }
166.        void BackHope()    {
167.            for (int p = 0; p < row; p++)        {
168.                for (int q = 0; q < column; q++)            {
169.                    if (dummyMap[p, q] == 3)                {
170.                        map[p, q] = 3;
171.                    }
172.                }
173.            }
174.        }
175.        int CheckRightLeft(int p, int q)    {
176.            if (q == 0 || q == column - 1)        {
177.                return 1;
178.            }
179.            else if (q - 1 >= 0)        {
180.                if (map[p, q - 1] == 2 || map[p, q - 1] == 6)
             {
181.                    return 1;
182.                }
183.            }
184.            else if (q + 1 < column)        {
```

```
185.                 if (map[p, q + 1] == 2 || map[p, q + 1] ==
     6)              {
186.                     return 1;
187.                 }
188.             }
189.             return 0;
190.         }
191.         int CheckTopBottom(int p, int q)     {
192.             if (p == 0 || p == row - 1)         {

193.                 return 1;
194.             }
195.             else if (p - 1 >= 0)         {
196.                 if (map[p - 1, q] == 2 || map[p - 1, q] ==
     6)              {
197.                     return 1;
198.                 }
199.             }
200.             else if (p + 1 < row)         {
201.                 if (map[p + 1, q] == 2 || map[p + 1, q] ==
     6)              {
202.                     return 1;
203.                 }
204.             }
205.             return 0;
206.         }
207.         int HuntPoint()     {
208.             for (int p = 0; p < row; p++)         {

209.                 for (int q = 0; q < column; q++)             {

210.                     if (map[p, q] == 0)                 {

211.                         if ((p == 0 && q == 0) || (p == 0 && q
    == column - 1) || (p == row - 1 && q == 0) || (p == row - 1 &&
     q == column - 1))                     {
212.                             if (CheckTopBottom(p, q) == 1)
                    {
213.                                 rowPoint = p;

214.                                 columnPoint = q;

215.                                 return 1;
```

```
216.                             }
217.                         }
218.                         else if (p - 1 >= 0)
    {
219.                             if ((map[p - 1, q] == 6 && PlayerW
    ay == true) || (map[p - 1, q] == 5))                        {
220.                                 histRow[1] = p - 1;
221.                                 histCol[1] = q;
222.                                 direction = 3;
223.                                 rowPoint = p;
224.                                 columnPoint = q;
225.                                 PlayerWay = false;
226.                                 return 1;
227.                             }
228.                         }
229.                         else if (p + 1 < row)
     {
230.                             if ((map[p + 1, q] == 6 && PlayerW
    ay == true) || (map[p + 1, q] == 5))                        {
231.                                 histRow[1] = p + 1;
232.                                 histCol[1] = q;
233.                                 direction = 1;
234.                                 rowPoint = p;
235.                                 columnPoint = q;
236.                                 PlayerWay = false;
237.                                 return 1;
238.                             }
239.                         }
```

```
240.                         else if (q - 1 >= 0)
    {
241.                             if ((map[p, q - 1] == 6 && PlayerW
    ay == true) || (map[p, q - 1] == 5))                        {
242.                                 histRow[1] = p;
243.                                 histCol[1] = q - 1;
244.                                 direction = 2;
245.                                 rowPoint = p;
246.                                 columnPoint = q;
247.                                 PlayerWay = false;
248.                                 return 1;
249.                             }
250.                         }
251.                         else if (q + 1 < column)
        {
252.                             if ((map[p, q + 1] == 6 && PlayerW
    ay == true) || (map[p, q + 1] == 5))                        {
253.                                 histRow[1] = p;
254.                                 histCol[1] = q + 1;
255.                                 direction = 4;
256.                                 rowPoint = p;
257.                                 columnPoint = q;
258.                                 PlayerWay = false;
259.                                 return 1;
260.                             }
261.                         }
262.                     }
263.                 }
264.             }
```

```
265.            return  0;
266.        }
267.        void  FirstHunt()     {
268.            rowPoint  =  Random.Range(0,  row  -  1);
269.            columnPoint  =  Random.Range(0,  column  -  1);
270.            frow  =  rowPoint;
271.            fcolumn  =  columnPoint;
272.            dummyMap[frow,  fcolumn]  =  6;
273.        }
274.        void  DoHunt()     {
275.            int  huntStatus  =  0;
276.            if (first)         {
277.                FirstHunt();
278.                first  =  false;
279.            }
280.            else         {
281.                huntStatus  =  HuntPoint();
282.                if  (huntStatus  ==  0)             {

283.                    isHunt  =  false;
284.                }
285.            }
286.            isKill  =  true;
287.            while  (isKill)         {
288.                DoKill();
289.            }
290.        }
291.        void  DoKill()     {
292.            histRow[0]  =  histRow[1];
293.            histCol[0]  =  histCol[1];
294.            WayObserve();
295.            if (character ==  true)         {
296.                map[rowPoint,  columnPoint]  =  6;
297.                character  =  false;
298.            }
299.            else         {
300.                map[rowPoint,  columnPoint]  =  1;
301.            }
302.            if  (count <= 0)         {
303.                isKill  =  false;
304.            }
305.            else  if (count > 0)         {
306.                hisDir  =  direction;
```

```
307.                 direction = randomDirection[Random.Range(0, count
    - 1)];
308.                 histRow[1] = rowPoint;
309.                 histCol[1] = columnPoint;
310.                 Movekill();
311.             }
312.         }
313.         int CheckTurn(int nrow, int ncol)     {
314.             if (direction == 0)         {
315.                 return 1;
316.             }
317.             else if ((histRow[0] == -1 && histCol[0] == -
    1) && (histRow[1] != -1 && histCol[1] != -
    1))         {
318.                 return 1;
319.             }
320.             else if (nrow != histRow[0] && ncol != histCol[0] )
           {
321.                 return 1;
322.             }
323.             return 0;
324.         }
325.         void ObstacleMark(int i, int j)     {
326.             if (future)         {
327.                 obs[obstacle].x = i;
328.                 obs[obstacle].y = j;
329.                 obstacle++;
330.             }
331.         }
332.         void Movekill()     {
333.             if (direction == 1)         {
334.                 if (CheckTurn(rowPoint - 1, columnPoint) == 1 &
    & hisDir != 0)             {
335.                     map[rowPoint, columnPoint] = 5;

336.                     if ((hisDir == 2 || hisDir == 0) && colum
    nPoint + 1 < column)                 {
337.                         ObstacleMark(rowPoint, columnPoint + 1);

338.                         map[rowPoint, columnPoint + 1] = 2;

339.                     }
```

```
340.                     else if ((hisDir == 4 || hisDir == 0) &&
    columnPoint - 1 >= 0)                {
341.                         ObstacleMark(rowPoint, columnPoint - 1);

342.                         map[rowPoint, columnPoint - 1] = 2;

343.                     }
344.                 }
345.                 rowPoint--;
346.             }
347.             else if (direction == 2)         {
348.                 if (CheckTurn(rowPoint, columnPoint + 1) == 1 &
    & hisDir != 0)             {
349.                     map[rowPoint, columnPoint] = 5;

350.                     if ((hisDir == 3 || hisDir == 0) && rowPo
    int + 1 < row)                 {
351.                         ObstacleMark(rowPoint + 1, columnPoint);

352.                         map[rowPoint + 1, columnPoint] = 2;

353.                     }
354.                     else if ((hisDir == 1 || hisDir == 0) &&
    rowPoint - 1 >= 0)                 {
355.                         ObstacleMark(rowPoint - 1, columnPoint);

356.                         map[rowPoint - 1, columnPoint] = 2;

357.                     }
358.                 }
359.                 columnPoint++;
360.             }
361.             else if (direction == 3)         {
362.                 if (CheckTurn(rowPoint + 1, columnPoint) == 1 &
    & hisDir != 0)             {
363.                     map[rowPoint, columnPoint] = 5;

364.                     if ((hisDir == 2 || hisDir == 0) && colum
    nPoint + 1 < column)                 {
365.                         ObstacleMark(rowPoint, columnPoint + 1);

366.                         map[rowPoint, columnPoint + 1] = 2;
```

```
367.                     }
368.                     else if ((hisDir == 4 || hisDir == 0) &&
    columnPoint - 1 >= 0)                 {
369.                         ObstacleMark(rowPoint, columnPoint - 1);

370.                         map[rowPoint, columnPoint - 1] = 2;

371.                     }
372.                 }
373.                 rowPoint++;
374.             }
375.             else if (direction == 4)         {
376.                 if (CheckTurn(rowPoint, columnPoint - 1) == 1 &
    & hisDir != 0)             {
377.                     map[rowPoint, columnPoint] = 5;

378.                     if ((hisDir == 3 || hisDir == 0) && rowPo
    int + 1 < row)                 {
379.                         ObstacleMark(rowPoint + 1, columnPoint);

380.                         map[rowPoint + 1, columnPoint] = 2;

381.                     }
382.                     else if ((hisDir == 1 || hisDir == 0) &&
    rowPoint - 1 >= 0)                 {
383.                         ObstacleMark(rowPoint - 1, columnPoint);

384.                         map[rowPoint - 1, columnPoint] = 2;

385.                     }
386.                 }
387.                 columnPoint--;
388.             }
389.         }
390.         void WayObserve()     {
391.             int available = 0;
392.             count = 0;
393.             randomDirection = new int[4];
394.             if (rowPoint != 0)         {
395.                 if (CheckTurn(rowPoint - 1, columnPoint) == 1 &
    & (map[rowPoint - 1, columnPoint] == 0 || map[rowPoint - 1,
    columnPoint] == 3))             {
```

```
396.                     if ((direction == 2 || direction == 0) &&
    columnPoint + 1 < column)                {
397.                         if (map[rowPoint, columnPoint + 1] == 0
    || map[rowPoint, columnPoint + 1] == 2)                    {

398.                             randomDirection[available] = 1;

399.                             available++;
400.                         }
401.                         else if (columnPoint + 2 < column)
                {
402.                             if (map[rowPoint, columnPoint + 2] ==
    1 && (map[rowPoint, columnPoint + 1] == 0 || map[rowPoint, co
    lumnPoint + 1] == 2))                        {

403.                                 randomDirection[available] = 1;

404.                                 available++;
405.                             }
406.                         }
407.                     }
408.                     else if ((direction == 4 || direction == 0
    ) && columnPoint - 1 >= 0)                {

409.                         if (map[rowPoint, columnPoint - 1] == 0
    || map[rowPoint, columnPoint - 1] == 2)                    {

410.                             randomDirection[available] = 1;

411.                             available++;
412.                         }
413.                         else if (columnPoint - 2 >= 0 && (map[
    rowPoint, columnPoint - 1] == 0 || map[rowPoint, columnPoint -
     1] == 2))                    {
414.                             if (map[rowPoint, columnPoint - 2] ==
     1)                        {
415.                                 randomDirection[available] = 1;

416.                                 available++;
417.                             }
418.                         }
419.                     }
```

```
420.                    else if ((columnPoint == 0 || columnPoint ==
     column - 1) && (map[rowPoint - 1, columnPoint] == 0 || map[r
   owPoint - 1, columnPoint] == 3))                {
421.                        randomDirection[available] = 1;
422.                        available++;
423.                    }
424.                }
425.                else            {
426.                    if (map[rowPoint - 1, columnPoint] == 0 ||
     map[rowPoint - 1, columnPoint] == 3)                {
427.                        randomDirection[available] = 1;
428.                        available++;
429.                    }
430.                }
431.            }
432.            if (columnPoint != column - 1)         {
433.                if (CheckTurn(rowPoint, columnPoint + 1) == 1 &
   & (map[rowPoint, columnPoint + 1] == 0 || map[rowPoint, column
   Point + 1] == 3))            {
434.                    if ((direction == 3 || direction == 0) &&
     rowPoint + 1 < row)                {
435.                        if (map[rowPoint + 1, columnPoint] == 0
     || map[rowPoint + 1, columnPoint] == 2)                    {
436.                            randomDirection[available] = 2;
437.                            available++;
438.                        }
439.                        else if (rowPoint + 2 < row && (map[ro
   wPoint + 1, columnPoint] == 0 || map[rowPoint + 1, columnPoin
   t] == 2))                   {
440.                            if (map[rowPoint + 2, columnPoint] ==
      1)                        {
441.                                randomDirection[available] = 2;
442.                                available++;
443.                            }
444.                        }
445.                    }
```

```
446.                     else if ((direction == 1 || direction == 0
    ) && rowPoint - 1 >= 0)                 {
447.                         if (map[rowPoint - 1, columnPoint] == 0
     || map[rowPoint - 1, columnPoint] == 0)                     {
448.                             randomDirection[available] = 2;
449.                             available++;
450.                         }
451.                         else if (rowPoint - 2 >= 0 && (map[rowP
    oint - 1, columnPoint] == 0 || map[rowPoint - 1, columnPoint]
      == 0))                     {
452.                             if (map[rowPoint - 2, columnPoint] ==
     1)                         {
453.                                 randomDirection[available] = 2;
454.                                 available++;
455.                             }
456.                         }
457.                     }
458.                     else if ((rowPoint == 0 || rowPoint == row - 1)
     && (map[rowPoint, columnPoint + 1] == 0 || map[rowPoint, col
    umnPoint + 1] == 3))                 {
459.                         randomDirection[available] = 2;
460.                         available++;
461.                     }
462.                 }
463.                 else            {
464.                     if (map[rowPoint, columnPoint + 1] == 0 ||
     map[rowPoint, columnPoint + 1] == 3)                 {
465.                         randomDirection[available] = 2;
466.                         available++;
467.                     }
468.                 }
469.             }
470.             if (rowPoint != row - 1)        {
471.                 if (CheckTurn(rowPoint + 1, columnPoint) == 1 &
    & (map[rowPoint + 1, columnPoint] == 0 || map[rowPoint + 1,
    columnPoint] == 3))             {
```

```
472.                     if ((direction == 2 || direction == 0) &&
    columnPoint + 1 < column)                 {
473.                         if (map[rowPoint, columnPoint + 1] == 0
    || map[rowPoint, columnPoint + 1] == 2)                     {

474.                             randomDirection[available] = 3;

475.                             available++;
476.                         }
477.                         else if (columnPoint + 2 < column && (
    map[rowPoint, columnPoint + 1] == 0 || map[rowPoint, columnPoin
    t + 1] == 2))                     {
478.                             if (map[rowPoint, columnPoint + 2] ==
     1)                         {
479.                                 randomDirection[available] = 3;

480.                                 available++;
481.                             }
482.                         }
483.                     }
484.                     else if ((direction == 4 || direction == 0
    ) && columnPoint - 1 >= 0)                 {
485.                         if (map[rowPoint, columnPoint - 1] == 0
    || map[rowPoint, columnPoint - 1] == 2)                     {

486.                             randomDirection[available] = 3;

487.                             available++;
488.                         }
489.                         else if (columnPoint - 2 >= 0 && (map[
    rowPoint, columnPoint - 1] == 0 || map[rowPoint, columnPoint -
     1] == 2))                     {
490.                             if (map[rowPoint, columnPoint - 2] ==
     1)                         {
491.                                 randomDirection[available] = 3;

492.                                 available++;
493.                             }
494.                         }
495.                     }
496.                     else if ((columnPoint == 0 || columnPoint == colu
    mn - 1) && (map[rowPoint + 1, columnPoint] == 0 || map[rowPoin
    t + 1, columnPoint] == 0))                 {
```

```
497.                         randomDirection[available] = 3;

498.                         available++;
499.                     }
500.                 }
501.                 else            {
502.                     if (map[rowPoint + 1, columnPoint] == 0 ||
    map[rowPoint + 1, columnPoint] == 3)                {

503.                         randomDirection[available] = 3;

504.                         available++;
505.                     }
506.                 }
507.             }
508.             if (columnPoint != 0)         {
509.                 if (CheckTurn(rowPoint, columnPoint - 1) == 1 &
    & (map[rowPoint, columnPoint - 1] == 0 || map[rowPoint, column
    Point - 1] == 3))            {
510.                     if ((direction == 3 || direction == 0) &&
    rowPoint + 1 < row)                {
511.                         if (map[rowPoint + 1, columnPoint] == 0
    || map[rowPoint + 1, columnPoint] == 2)                    {

512.                             randomDirection[available] = 4;

513.                             available++;
514.                         }
515.                         else if (rowPoint + 2 < row && (map[ro
    wPoint + 1, columnPoint] == 0 || map[rowPoint + 1, columnPoin
    t] == 2))                   {
516.                             if (map[rowPoint + 2, columnPoint] ==
     1)                         {
517.                                 randomDirection[available] = 4;

518.                                 available++;
519.                             }
520.                         }
521.                     }
522.                     else if ((direction == 1 || direction == 0
    ) && rowPoint - 1 >= 0)                {
```

```
523.                         if (map[rowPoint - 1, columnPoint] == 0
    || map[rowPoint - 1, columnPoint] == 2)                    {
524.                             randomDirection[available] = 4;
525.                             available++;
526.                         }
527.                         else if (rowPoint - 2 >= 0 && (map[row
   Point - 1, columnPoint] == 0 || map[rowPoint - 1, columnPoint
   ] == 2))                    {
528.                             if (map[rowPoint - 2, columnPoint] ==
     1)                         {
529.                                 randomDirection[available] = 4;
530.                                 available++;
531.                             }
532.                         }
533.                     }
534.                     else if ((rowPoint == 0 || rowPoint == row - 1)
   && (map[rowPoint, columnPoint - 1] == 0 || map[rowPoint, colum
   nPoint - 1] == 3))                {
535.                         randomDirection[available] = 4;
536.                         available++;
537.                     }
538.                 }
539.                 else            {
540.                     if (map[rowPoint, columnPoint - 1] == 0 ||
    map[rowPoint, columnPoint - 1] == 3)                {
541.                         randomDirection[available] = 4;
542.                         available++;
543.                     }
544.                 }
545.             }
546.             count = available;
547.         }
548.     }
```

**Kode Sumber A.1 Hunt-and-Kill**

# LAMPIRAN B

**Gambar B.1 Stage 2**

**Gambar B.2 Stage 3-1**

**Gambar B.3 Stage 3-2**

**Gambar B.4 Stage 4-1**

**Gambar B.5 Stage 4-2**

**Gambar B.6 Stage 4-3**

**Gambar B.7 Stage 5-1**

**Gambar B.8 Stage 5-2**

**Gambar B.9 Stage 5-3**

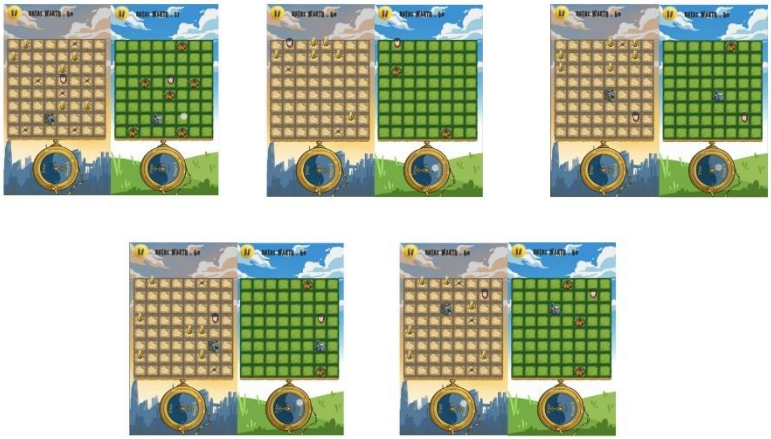
**Gambar B.10 Stage 5-4**

**Gambar B.11 Stage 6-1**

**Gambar B.12 Stage 6-2**

**Gambar B.13 Stage 6-3**

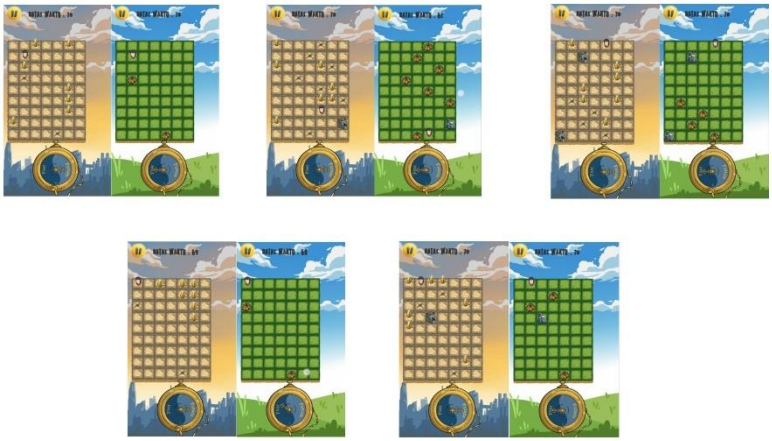
**Gambar B.14 Stage 6-4**



**Gambar B.15 Stage 6-5**

**Gambar B.16 Stage 7-1**

**Gambar B.17 Stage 7-2**



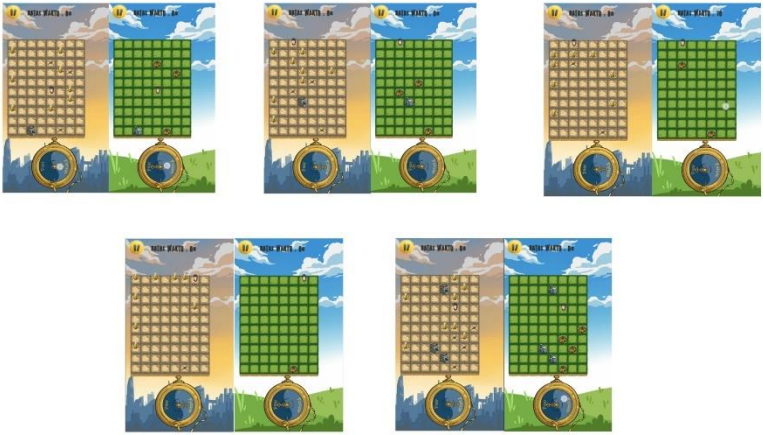
**Gambar B.18 Stage 7-3**

**Gambar B.19 Stage 7-4**

**Gambar B.20 Stage 7-5**

**Gambar B.21 Stage 8-1**

**Gambar B.22 Stage 8-2**



**Gambar B.23 Stage 8-3**

**Gambar B.24 Stage 8-4**

**Gambar B.25 Stage 8-5**

**Gambar B.26 Stage 9-1**

**Gambar B.27 Stage 9-2**

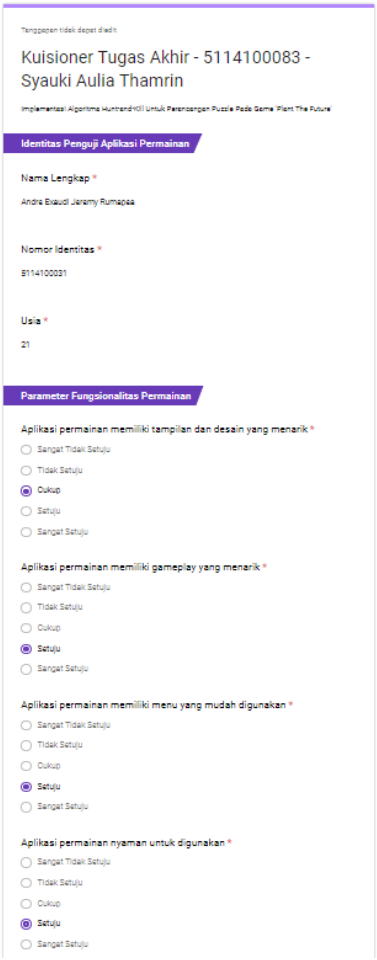
**Gambar B.28 Stage 9-3**

**Gambar B.29 Stage 9-4**

**Gambar B.30 Stage 9-5**

*[Halaman ini sengaja dikosongkan]*

# LAMPIRAN C



Tanggapan tidak dapat diedit

Kuisioner Tugas Akhir - 5114100083 - Syauki Aulia Thamrin

Implementasi Algoritma Huntrand'nOS Untuk Perancangan Puzzle Pada Game Plant The Future

**Identitas Penguji Aplikasi Permainan**

Nama Lengkap *

Andre Ekaudi Jeremy Rumapea

Nomor Identitas *

5114100031

Usia *

21

**Parameter Fungsionalitas Permainan**

Aplikasi permainan memiliki tampilan dan desain yang menarik *

- Sangat Tidak Setuju
- Tidak Setuju
- **Cukup**
- Setuju
- Sangat Setuju

Aplikasi permainan memiliki gameplay yang menarik *

- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- **Setuju**
- Sangat Setuju

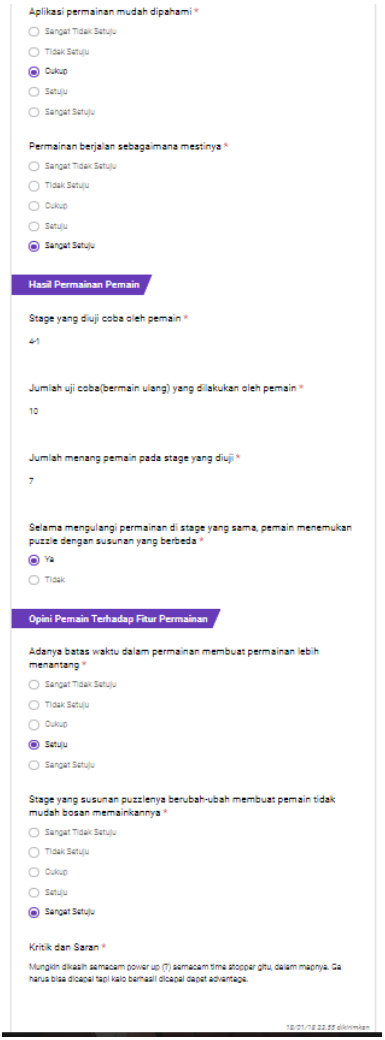
Aplikasi permainan memiliki menu yang mudah digunakan *

- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- **Setuju**
- Sangat Setuju

Aplikasi permainan nyaman untuk digunakan *

- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- **Setuju**
- Sangat Setuju

**Gambar C.1 Form pengujian pengguna 1.A**



Aplikasi permainan mudah dipahami *
- Sangat Tidak Setuju
- Tidak Setuju
- (•) Cukup
- Setuju
- Sangat Setuju

Permainan berjalan sebagaimana mestinya *
- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- Setuju
- (•) Sangat Setuju

**Hasil Permainan Pemain**

Stage yang diuji coba oleh pemain *

4-1

Jumlah uji coba(bermain ulang) yang dilakukan oleh pemain *

10

Jumlah menang pemain pada stage yang diuji *

7

Selama mengulangi permainan di stage yang sama, pemain menemukan puzzle dengan susunan yang berbeda *
- (•) Ya
- Tidak

**Opini Pemain Terhadap Fitur Permainan**

Adanya batas waktu dalam permainan membuat permainan lebih menantang *
- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- (•) Setuju
- Sangat Setuju

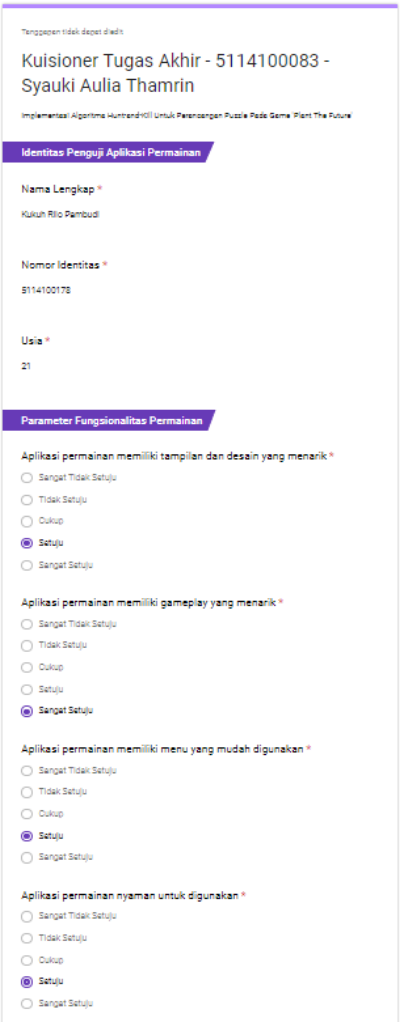
Stage yang susunan puzzlenya berubah-ubah membuat pemain tidak mudah bosan memainkannya *
- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- Setuju
- (•) Sangat Setuju

Kritik dan Saran *

Mungkin dikasih semacam power up (?) semacam time stopper gitu, dalam mapnya. Ga harus bisa dicapai tapi kalo berhasil dicapai dapet advantage.

18/01/19 23.55 dikirimkan

**Gambar C.2 Form pengujian pengguna 1.B**



Tanggapan tidak dapat diedit

Kuisioner Tugas Akhir - 5114100083 - Syauki Aulia Thamrin

Implementasi Algoritma Huntandvill Untuk Perancangan Puzzle Pada Game Plant The Future

**Identitas Penguji Aplikasi Permainan**

Nama Lengkap *

Kukuh Rilo Pambudi

Nomor Identitas *

5114100178

Usia *

21

**Parameter Fungsionalitas Permainan**

Aplikasi permainan memiliki tampilan dan desain yang menarik *

- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- **Setuju**
- Sangat Setuju

Aplikasi permainan memiliki gameplay yang menarik *

- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- Setuju
- **Sangat Setuju**

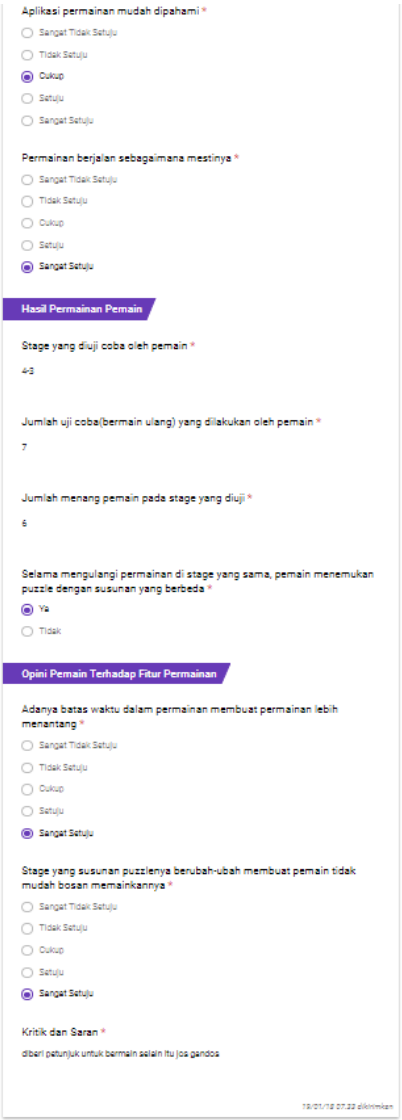
Aplikasi permainan memiliki menu yang mudah digunakan *

- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- **Setuju**
- Sangat Setuju

Aplikasi permainan nyaman untuk digunakan *

- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- **Setuju**
- Sangat Setuju

**Gambar C.3 Form pengujian pengguna 2.A**



Aplikasi permainan mudah dipahami *
- Sangat Tidak Setuju
- Tidak Setuju
- (•) Cukup
- Setuju
- Sangat Setuju

Permainan berjalan sebagaimana mestinya *
- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- Setuju
- (•) Sangat Setuju

**Hasil Permainan Pemain**

Stage yang diuji coba oleh pemain *

4-3

Jumlah uji coba(bermain ulang) yang dilakukan oleh pemain *

7

Jumlah menang pemain pada stage yang diuji *

6

Selama mengulangi permainan di stage yang sama, pemain menemukan puzzle dengan susunan yang berbeda *
- (•) Ya
- Tidak

**Opini Pemain Terhadap Fitur Permainan**

Adanya batas waktu dalam permainan membuat permainan lebih menantang *
- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- Setuju
- (•) Sangat Setuju

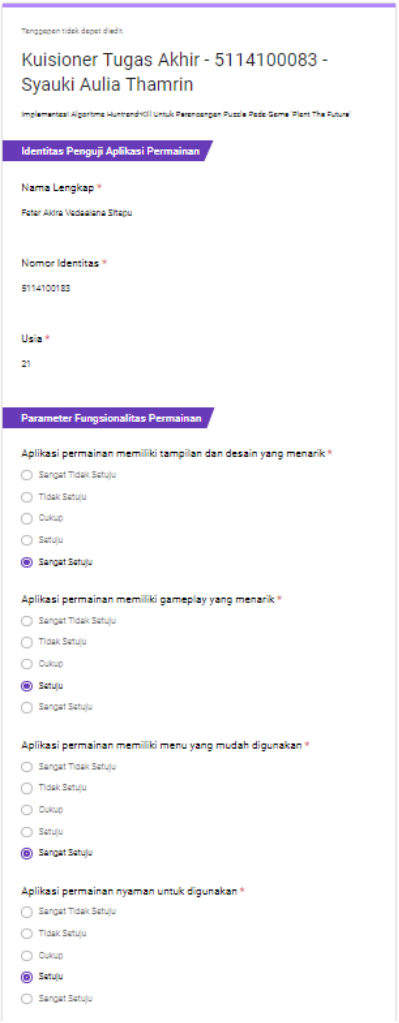
Stage yang susunan puzzlenya berubah-ubah membuat pemain tidak mudah bosan memainkannya *
- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- Setuju
- (•) Sangat Setuju

Kritik dan Saran *

diberi petunjuk untuk bermain selain itu jos gandos

19/01/19 07.22 dikirimkan

**Gambar C.4 Form pengujian pengguna 2.B**



Tanggapan tidak dapat diedit

Kuisioner Tugas Akhir - 5114100083 - Syauki Aulia Thamrin

Implementasi Algoritma HuntrendHill Untuk Perancangan Puzzle Pada Game Plant The Future

**Identitas Penguji Aplikasi Permainan**

Nama Lengkap *

Fetar Akira Vedaalana Sitepu

Nomor Identitas *

5114100183

Usia *

21

**Parameter Fungsionalitas Permainan**

Aplikasi permainan memiliki tampilan dan desain yang menarik *

- ○ Sangat Tidak Setuju
- ○ Tidak Setuju
- ○ Cukup
- ○ Setuju
- ◉ Sangat Setuju

Aplikasi permainan memiliki gameplay yang menarik *

- ○ Sangat Tidak Setuju
- ○ Tidak Setuju
- ○ Cukup
- ◉ Setuju
- ○ Sangat Setuju

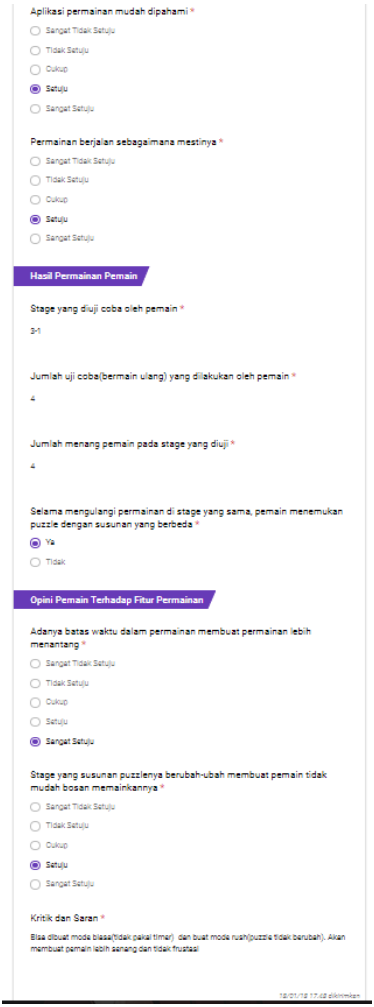
Aplikasi permainan memiliki menu yang mudah digunakan *

- ○ Sangat Tidak Setuju
- ○ Tidak Setuju
- ○ Cukup
- ○ Setuju
- ◉ Sangat Setuju

Aplikasi permainan nyaman untuk digunakan *

- ○ Sangat Tidak Setuju
- ○ Tidak Setuju
- ○ Cukup
- ◉ Setuju
- ○ Sangat Setuju

**Gambar C.5 Form pengujian pengguna 3.A**



Aplikasi permainan mudah dipahami *
- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- Setuju
- Sangat Setuju

Permainan berjalan sebagaimana mestinya *
- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- Setuju
- Sangat Setuju

**Hasil Permainan Pemain**

Stage yang diuji coba oleh pemain *
3-1

Jumlah uji coba(bermain ulang) yang dilakukan oleh pemain *
4

Jumlah menang pemain pada stage yang diuji *
4

Selama mengulangi permainan di stage yang sama, pemain menemukan puzzle dengan susunan yang berbeda *
- Ya
- Tidak

**Opini Pemain Terhadap Fitur Permainan**

Adanya batas waktu dalam permainan membuat permainan lebih menantang *
- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- Setuju
- Sangat Setuju

Stage yang susunan puzzlenya berubah-ubah membuat pemain tidak mudah bosan memainkannya *
- Sangat Tidak Setuju
- Tidak Setuju
- Cukup
- Setuju
- Sangat Setuju

Kritik dan Saran *

Bisa dibuat mode biasa(tidak pakai timer) dan buat mode rush(puzzle tidak berubah). Akan membuat pemain lebih senang dan tidak frustasi

12/01/19 17.02 dikirimkan

**Gambar C.6 Form pengujian pengguna 3.B**