PEMBUATAN GAME REAL TIME STRATEGY BERBASIS ANATOMI TUBUH MANUSIA MENGGUNAKAN UNITY 3D

TUGAS AKHIR
Diajukan untuk Memenuhi Salah Satu Syarat Kelulusan
Program Pendidikan Sarjana

Oleh :
Yonatan Nathanael
2015130039

JURUSAN INFORMATIKA
SEKOLAH TINGGI MANAJEMEN INFORMATIKA & KOMPUTER – LIKMI
BANDUNG
2021

PEMBUATAN GAME REAL TIME STRATEGY BERBASIS ANATOMI TUBUH MANUSIA MENGGUNAKAN UNITY 3D

Oleh :
Yonatan Nathanael
2015130039

Bandung, 5 Maret 2021
Menyetujui,

Rachmat Selamat, S.Kom., M.T.	Dhanny Setiawan, S.T., M.T.
Pembimbing	Ketua Jurusan

JURUSAN INFORMATIKA
SEKOLAH TINGGI MANAJEMEN INFORMATIKA & KOMPUTER – LIKMI
BANDUNG
2021

ABSTRAK

Game merupakan media hiburan bagi banyak orang, dan banyak *game* sekarang yang menguji adaptasi dan tingkat kecepatan berfikir. Contoh *game* yang memaksa pemain untuk adaptasi dengan situasi dan harus berfikir cepat adalah game *real time strategy*. *Game real time strategy* membawa pemain untuk berfikir ke depan.

Cells merupakan *game* yang memiliki genre *real time strategy* dimana pemain akan menggerakan sekelompok pasukan *cell*, melatih pasukan *cell*, membangun bangunan, mengumpulkan resource. Menggerakan prajurit di dalam medan *game* unuk mengalahkan musuh yang menyerang tubuh manusia. Dan memberi informasi tentang *cell* tubuh manusia dan musuh yang menyerang. Dengan menambahkan informasi *cells*, peta, dan kontrol pasukan *game*, dirancang dan di *compile* menjadi .exe yang akan di buka di platform PC. Unity di pilih untuk menjadi *game engine Cells* karena dapat membantu membuat *game real time strategy* dengan menggunakan navigasi yang di miliki oleh *engine* unity.

Penyusunan tugas akhir ini dilakukan melalui studi pustaka terhadap beberapa refrensi seperti buku, jurnal, dan situs resmi mengenai judul penelitian yang terkait dengan *game Cells* genre *real time strategy*. Dalam pengembangan ini menggunakan metode *prototyping* dan menggunakan pemograman bahasa C#.

Diagram UML yang digunakan dalam pembuatan game ini adalah *Use Case Diagram*, *Scenario Diagram*, *Activity Diagram*, dan *Class Diagram*. Alat alat yang digunakan dari unity adalah *GameObject*, *AI NavMesh*, *Collider*, *Animator*, *Sprite Renderer*, *Camera*, *Canvas*, *TextMeshPro*, dan *BinaryFormatter*.

KATA PENGANTAR

Puji syukur panjatkan ke hadirat Tuhan Yang Maha Esa, karena anugerah dan tuntunanNya, akhirnya dapat menyelesaikan perkulihan hingga penyusunan tuhas akhir. Banyak pengalaman, serta pelajaran yang dapat dipetik selama perkulihan hingga mencapai tugas akhir di STMIK LIKMI.

Dengan segala usaha yang telah dilakukan dan menyadari banyak sekali kekurangan dari rancangan, dan isi dari tugas akhir. Untuk itu memohon minta maaf atas kekurangan yang terdapat di dalam perancangan tugas akhir ini. Dengan harapan yang besar, semuga tugas akhir ini memiliki manfaat yang dapat dibagikan kepada pembaca tugas akhir ini.

Pembantuan penulisan tugas akhir ini juga tidak lepas dari bantuan barbagai pihak, khususnya para dosen, keluarga, kerabatan, rekan mahasiswa. Dalam kesempatan ini ingin mengucapkan terima kasih yang sangat besar kepada:

1. Bapa Rachmat Selamat, S.Kom., M.T. selaku dosen pembimbing yang telah memberi dukungan, arahan, ide, serta semangat selama pengerjaan tugas akhir ini.
2. Bapak Dhanny Setiawan, S.T., M.T. selaku ketua jurusan Teknik Informatika STMIK LIKMI.
3. Seluruh dosen dan staff STMIK LIKMI yang telah membantu agar penulisan tugas akhir ini dapat selesai dengan lancar.
4. Terutama kepada orang tua yang selalu mendoaakan, memberi motivasi, dukungan, dan pengorbanan yang sangat besar sampai menyelesaikan tugas akhir ini.
5. Seluruh rekan seperjuang, khususnya semua rekan Angkatan 2015 di STMIK LIKMI yang telah membantu dalam menyusun tugas akhir ini yang tidak bisa disebutkan satu persatu, terima kasih memberikan arahan dan bantuan.
6. Seluruh rekan online yang telah memembantu menguji permainan yang dibuat walau sebagai hobi sebelum mengerjakan tugas akhir.

Menyadari bahwa tugas akhir ini masih jauh dari kesempurnaan. Karena itu, dibukanya menerima setiap saran dan kritik untuk membangun tugas akhir ini.

Akhir kata, diharapkan bahwa tugas akhir ini dapat memberikan ilmu dan manfaat dalam menambah wawasan dan ide bagi para pembaca.

Bandung, 5 Maret 2021

Penulis

DAFTAR ISI

ABSTRAK.....i

KATA PENGANTAR ii

DAFTAR ISI.....iv

DAFTAR GAMBAR vii

DAFTAR TABEL ix

DAFTAR SIMBOL xi

DAFTAR LAMPIRAN xiii

BAB I PENDAHULUAN.....1

1.1. Latar Belakang 1

1.2. Rumusan Masalah 2

1.3. Tujuan 2

1.4. Batas Masalah 2

1.5. Kegunaan Hasil.....2

1.6. Sistematika Penulisan.....2

BAB II LANDASAN TEORI.....4

2.1. Rekayasa Perangkat Lunak.....4

2.1.1. Definisi Rekayasa Perangkat Lunak.....4

2.1.2. *Object Oriented Programming* 4

2.1.2.1. Sejarah *Object Oriented Programming* 5

2.1.2.2. Karakteristik dari *Object Oriented Programing*.....5

2.1.3. Model Pengembangan *Prototyping* 6

2.2. UML.....7

2.3. Definisi *Game*.....11

2.3.1. Genre *Games*.....12

2.3.2. Sejarah Pendek *Game Real Time Strategy*.....16

2.3.3. Definisi *Game Real Time Strategy*.....17

2.3.4. Ciri *Game Real Time Strategy*.....18

2.4. Unity & NavMesh 18
2.5. Bahasa Pemograman C# 23
2.6. Biologi 24
2.6.1. Immune System 24
2.6.2. Attacking Microbes 33
BAB III ANALISIS DAN PERANCANGAN 35
3.1. Gambaran Umum Perangkat Lunak 35
3.1.1. Kontrol Pemain 36
3.1.2. Jenis Cells 37
3.1.3. Spesifikasi Kebutuhan Perangkat Lunak 39
3.1.4. Spesifikasi Kebutuhan Perangkat Keras 39
3.1.5. Analisi Kebutuhan Fungsional 40
3.2. *Use Case Diagram* 40
3.3. *Scenario Diagram* 41
3.3.1. *Scenario Play Game* 41
3.3.2. *Scenario Library* 48
3.3.3. *Scenario Scores* 48
3.3.4. *Scenario Setting* 49
3.4. *Activity Diagram* 50
3.4.1. *Activity Diagram Play Game* 50
3.4.2. *Activity Diagram Library* 52
3.4.3. *Activity Diagram Scores* 53
3.4.4. *Activity Diagram Setting* 53
3.5. *Class Diagram* 54
3.6. Rancangan Antarmuka 56
BAB 4 IMPLEMENTASI DAN PENGUJIAN 59
4.1. Spesifikasi Kebutuhan Perangkat Keras 59
4.2. Penjelasan *Menu* 59
4.3. Pengujian Antar Muka 60

4.3.1. Antar Muka *Main Menu* 60
4.3.2. Antar Muka *Library* 61
4.3.3. Antar Muka *Scores* 61
4.3.4. Antar Muka *Setting* 62
4.3.5. Antar Muka *Game* 62
4.3.6. Antar Muka *Menu Paused* 63
4.3.7. Antar Muka *Game Over* 63
4.4. Pengujian Fungsi 64
BAB 5 KESIMPULAN DAN SARAN 68
5.1. Kesimpulan 68
5.2. Saran 68
DAFTAR PUSTAKA 69

DAFTAR GAMBAR

Gambar 2.1 Model *Prototyping* 7

Gambar 2.2 *Bone Marrow* 25

Gambar 2.3 *Trombocytes* 26

Gambar 2.4 *Red Blood Cell* 26

Gambar 2.5 *White Blood Cell* 27

Gambar 2.6 *Macrophage* 28

Gambar 2.7 *Eosinophils* 28

Gambar 2.8 *Lymph Node* 29

Gambar 2.9 *B Cell* 29

Gambar 2.10 *T Cell* 30

Gambar 2.11 *Nervous System* 31

Gambar 2.12 *Intestine* 31

Gambar 2.13 *Artery* 32

Gambar 2.14 *Fat* 32

Gambar 2.15 *Streptococcus Pneumoniae* 33

Gambar 2.16 *Helicobacter Pylori* 33

Gambar 2.17 *Influenza* 34

Gambar 2.18 *Anisakis* 34

Gambar 3.1 *Use Case Diagram* Perangkat Lunak *Game Cells* 40

Gambar 3.2 *Activity Diagram* "*Start Game"* 51

Gambar 3.3 *Activity Diagram* "*Library"* 52

Gambar 3.4 *Activity Diagram* "*Scores"* 53

Gambar 3.5 *Activity Diagram* "*Setting"* 54

Gambar 3.6 *Class Diagram* Perangkat Lunak *Game Cells* 55

Gambar 3.7 Tampilan *Main Menu* 56

Gambar 3.8 Tampilan *Play Game* 56

Gambar 3.9 Tampilan Dalam *Game* 57

Gambar 3.10 Tampilan *Cell Library* 57
Gambar 3.11 Tampilan *Score* 57
Gambar 3.12 Tampilan *Setting* 58
Gambar 4.1 Tampilan *Main Menu* 60
Gambar 4.2 Tampilan *Library Menu* 61
Gambar 4.3 Tampilan *Scores Menu* 61
Gambar 4.4 Tampilan *Setting Menu* 62
Gambar 4.5 Tampilan *Game*.....62
Gambar 4.6 Tampilan *Menu Paused* 63
Gambar 4.7 Tampilan *Game Over*.....63

DAFTAR TABEL

Tabel 2.1 UML *Structure Diagram* 8

Tabel 2.2 UML *Behavior Diagrams*.....9

Tabel 3.1 *Shortcut Keyboard* 36

Tabel 3.2 Jenis *Cells*.....37

Tabel 3.3 *Scenario Use Case Play Game*: *Level* 0 *Tutorial* 41

Tabel 3.4 *Scenario* alternatif *use case Play Game* 1: *Level* 0 *Tutorial Unit* Kalah Semua 43

Tabel 3.5 *Scenario* alternatif *use case Play Game* 2: *Level* 0 *Tutorial Nerve Center* Hancur.....43

Tabel 3.6 *Scenario* alternatif *use case Play Game* 3: *Level* 0 *Tutorial* Keluar Dari *Game* 43

Tabel 3.7 *Scenario* alternatif *use case Play Game* 4: *Level* 0 *Tutorial* Mengubah *Setting Volume* 43

Tabel 3.8 *Scenario* alternatif *use case Play Game* 5: Bermain di *Level* 1 *Exterminate* 44

Tabel 3.9 *Scenario* alternatif *use case Play Game* 6: *Level* 1 *Exterminate Nerve Center* Hancur.....44

Tabel 3.10 *Scenario* alternatif *use case Play Game* 7: *Level* 1 *Exterminate* Keluar Dari *Game*.....44

Tabel 3.11 *Scenario* alternatif *use case Play Game* 8: *Level* 1 *Exterminate* Mengubah *Setting* 45

Tabel 3.12 *Scenario* alternatif *use case Play Game* 9: *Level* 2 *Defend* 45

Tabel 3.13 *Scenario* alternatif *use case Play Game* 10: *Level* 2 *Defend Nerve Center* Hancur.....46

Tabel 3.14 *Scenario* alternatif *use case Play Game* 11: Level 2 Defend Keluar Dari *Game* 46

Tabel 3.15 *Scenario* alternatif *use case Play Game* 12: *Level* 2 *Defend* Mengubah *Setting Volume* 46

Tabel 3.16 *Scenario* alternatif *use case Play Game* 13: *Level* 3 *Survival* 47

Tabel 3.17 *Scenario* alternatif *use case Play Game* 15: *Level* 3 *Survival* Keluar Dari *Game*.....47

Tabel 3.18 *Scenario* alternatif *use case Play Game* 16: *Level* 3 *Survival* Mengubah *Setting Volume* 48

Tabel 3.19 *Scenario Use Case Library* 48

Tabel 3.20 *Scenario* alternatif *use case Library* 1: Data Tidak Pernah di Akses.....48

Tabel 3.21 *Scenario Use Case View Scores* 49

Tabel 3.22 *Scenario Use Case* 49

Tabel 3.23 *Scenario* alternatif *use case Setting* 1: Keluar Dari *Setting* Tanpa *Apply* ...49

Tabel 4.1 Spesifikasi Perangkat Keras 59

Tabel 4.2 Pengujian scene *Main Menu*.....64

Tabel 4.3 Pengujian *Play Game* 64

Tabel 4.4 Pengujian Panel *Library*.....66

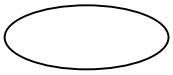
Tabel 4.5 Pengujian Panel *Scores*.....66

Tabel 4.6 Pengujian Panel *Setting*.....66

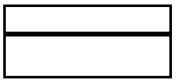
Tabel 4.7 Pengujian Panel *MenuPause*.....67

DAFTAR SIMBOL

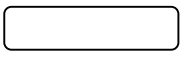
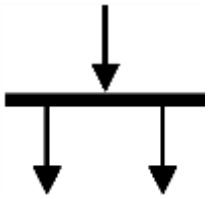
Use case diagram

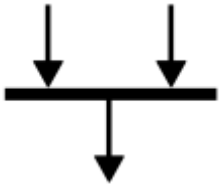
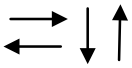
Nama Simbol	Simbol	Keterangan
Use Case		Sebuah kebiasaan atau fungsi utama (*key behaviour*) yang dilakukan oleh sistem perangkat lunak
Asosiasi		Hubungan antara aktor dengan *use case* yang menunjukan adanya interaksi aktor dengan sistem atau campur tangan aktor dalam suatu *use case*
Aktor		Entitas eksternal yang berhubungan dengan sistem

Class diagram

Nama Simbol	Simbol	Keterangan
Asosiasi		Menunjukan relasi yang mengindikasikan bahwa suatu kelas mereferensikan kelas yang lain.
Generalisasi		Hubungan dimana objek anak berbagi perilaku dan struktur data dari objek yang ada di atas objek induk
Class		Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.

Activity diagram

Nama Simbol	Simbol	Keterangan
Kondisi awal		Menunjukan kondisi awal kegiatan sistem
Kondisi akhir		Menunjukan kondisi berakhirnya kegiatan sistem
Action State		Kegiatan yang sistem lakukan
Fork		Kegiatan yang bercabang (boleh dilakukan salah satu atau semuanya sekaligus)

Nama Simbol	Simbol	Keterangan
Join		Kegiatan yang digabungkan
Desicion		Percabangan dimana sistem harus melakukan salah satu cabang keputusan
Arah aktivitas		Menunjukan arah sebuah *action state* ke *action state* selanjutnya

DAFTAR LAMPIRAN

Listing class PlayerControls 71
Listing class UnitStatus 82
Listing class BuildingStatus 89
Listing class Player 92
Listing class PlayerData 93
Listing class SaveScript 94
Listing class LibraryScript 95
Listing class SettingScript 98
Listing class SettingButtonScript 98
Listing class CameraMovement 99
Listing class ResourceManager 100
Listing class SystemNotifText 100
Listing class UIMovementButtonScript 101
Listing class GameAssets 102
Listing class MultipleUnitsButton 103
Listing class GlobalAttribute 103
Listing class DamagePopup 106
Listing class MissileScript 106
Listing class UIScript 108
Listing class UIBuildTrainButton 112
Listing class UITooltipScript 114
Listing class MapFogScript 116
Listing class EventTrigger 117

BAB I

PENDAHULUAN

1.1. Latar Belakang

Perkembangan *game* pada jaman ini mengalami perubahan yang cukup signifkan, baik dari segi grafis, *gameplay*, *in-game feature*, dan lain-lain. Hal tersebut menjadi salah satu daya tarik *user* untuk bermain *game*. Meskipun telah mengalami perkembangan yang cukup signifikan, tetap saja sebagian besar memiliki dampak negatif bagi para pemainnya, seperti kecanduan bermain *game*, menghabiskan uang yang seringkali jumlahnya tidak sedikit, menghabiskan waktu secara sia-sia yang membuat pemainnya menjadi tidak produktif, dan sebagainya. Faktanya, *game* juga memiliki dampak positif yang tidak diketahui oleh banyak orang, seperti melatih kemampuan berbahasa, melatih logika, melatih kemampuan spasial, pengenalan teknologi, kemampuan membaca, stimulasi otak, dan mengembangkan kemampuan imajinasi pemainnya. Terutama untuk jenis *game Real Time Strategy,* yang dimana pemain dituntut untuk dapat mengasah kemampuan berpikir selama permainan seperti *game* Starcraft 2, Warcraft 3, dan Age of Empire.

Oleh karena itu, membuat sebuah *Game Real Time Strategy* berbasis anatomi tubuh manusia sebagai bentuk visualisasi dan mengetahui bagaimana cara kerja tubuh manusia. Cara bermainya dengan menahan serangan bakteri yang akan menghancurkan tubuh manusia, yang keluar secara terus menerus selama jeda waktu tertentu. Pemain dapat membangun suatu bangunan, membentuk pertahanan, dan membuat sejumlah pasukan.

Perkembangan *Game Cells* menggunakan Unity, karena unity menggunakan C# dan banyak software juga menggunakan C#, Unity juga bagus untuk membuat *Game* 3D maupun 2D, dan Unity mendukung banyak *platform* seperti Windows, Mac, Linux, Mobile, dan lain lain.

Berdasarkan uraian di atas, maka diusulkan dalam pembuatan tugas akhir ini yang berjudul **"PEMBUATAN GAME REAL TIME STRATEGY BERBASIS ANATOMI TUBUH MANUSIA MENGGUNAKAN UNITY 3D".**

1.2. Rumusan Masalah

Masalah yang akan di bahas adalah, membuat aplikasi *game* strategi berbasis anatomi tubuh manusia menggunakan unity 3D.

1.3. Tujuan

Tujuan dari penulisan tugas akhir ini adalah sebagai berikut:

1. Sebagai salah satu syarat kelulusan pendidikan sarjana S1 informatika di perguruan tinggi STMIK LIKMI.
2. Menceritakan dan memberi edukasi bagaimana cara kerja *cell cell* tubuh manusia.
3. Melatih kempuan berfikir secara *real* time untuk pemain.

1.4. Batas Masalah

Batasan masalah dari *game* ini adalah sebagai berikut:

1. *Game* dimainkan *offline*.
2. *Game* dimainkan oleh *single player*.
3. *Platform* yang digunakan adalah Windows.

1.5. Kegunaan Hasil

Dengan adanya *game* ini diharapkan dapat mempermudah pemain mengenal lebih cara kerja dan mempelajari cell tubuh mereka sendiri, melatih kemampuan berfikir secara *real time*, dan sebagai salah satu sarana hiburan.

1.6. Sistematika Penulisan

Sistematika penulisan ini disusun dan dibagi kedalam 5 bab, meliputi hal-hal sebagai berikut ini:

BAB I PENDAHULUAN

Pada bab ini akan menjelaskan mengenai latar belakang, rumusan masalah, tujuan pembuatan, batasan masalah, kegunaan hasil, serta sistematika penulisan.

BAB II LANDASAN TEORI

Pada bab ini akan menjelaskan metode-metode pembuatan *Game Real Time Strategy* seperti metode perancangan *framework*, *pathfinding*, serta teori prototyping yang ada di dalam pembuatan *game*, *Unity engine*, dan tipe tipe cell yang ada di dalam *game*.

BAB III ANALISIS DAN PERANCANGAN PERANGKAT LUNAK

Pada bab ini berisi tentang analisis dan rancangan *Game Real Time Strategy* yang akan di buat. Metode yang digunakan dalam proses analisis dan perancangan meliputi pembuatan *use case digram, class diagram, activity diagram, scenario diagram, sequence diagram*, serta rancangan antar muka.

BAB IV IMPLEMENTASI DAN PENGUJIAN

Pada bab ini akan menguraikan mengenai implementasi dari perancangan *Game Real Time Strategy* dan membahas tentang hasil pengujian dari *Game* tersebut.

BAB V KESIMPULAN DAN SARAN

Pada Bab ini akan memberikan kesimpulan dan saran mengenai pembuatan *Game Real Time Strat*

BAB II

LANDASAN TEORI

2.1. Rekayasa Perangkat Lunak

2.1.1. Definisi Rekayasa Perangkat Lunak

Menurut Richard F. Schmidt, di dalam buku yang berjudul "*Software Engineering Architecture-Driven Software Development*", definisi rekayasa perangkat lunak adalah sebagai berikut:

"*System engineering is a discipline that meld interdisciplinary technical disciplines and project management practices to develop architectural design challenges associated with complex product development."* (Schmidt, 2013:1)

Berdasarkan deskripsi tersebut, dapat diartikan:

Rekayasa perangkat lunak adalah kedisiplinan yang membetuk teknik interdisipliner dan praktek manajemen proyek untuk mengembangkan tantangan desain arsitektur yang terkait dengan pengembangan produk yang kompleks.

Menurut Ian Sommerville, di dalam buku yang berjulul "*Software Engineering*, 9th *Edition*", definisi rekayasa perangkat lunak adalah sebagai berikut:

"*Software engineering is an engineering discipline that is concerned with all aspect of software production from the early stages of system specification through to maintaining the system after it has gone into use.*" (Sommerville, 2011:7)

Berdasarkan deskripsi tersebut, dapat diartikan:

Rekayasa perangkat lunak adalah teknik disiplin yang berkaitan dengan semua aspek dari perangkat lunak dari tahap awal spesifikasi sitem sampai mempertahankan sistemnya sesudah digunakan.

2.1.2. *Object Oriented Programming*

Object oriented programing menurut Dan Clark yang disertakan di dalam buku yang berjudul "*Beginning C# Object-Oriented Programming*" adalah sebagai berikut:

"Object oriented programing adalah sebuah pendekatan ke perkembangan perangkat lunak dimana strukturnya berdasarkan pada obyek yang berinteraksi sama satu yang lain

tuntuk membereskan tugas. Interaksi ini membutuhkan bentuk mengirim bolak balik antara obyek dan obyek. Dalam respon terhadap pesanan obyek dapat melakukan aksi atau metode." (Clark, 2011:1)

2.1.2.1. Sejarah *Object Oriented Programming*

Konsep dari *Object Oriented Programing* mulai muncul dari tengah 1960 dengan bahasa *programing* di panggil Simula, dan dikembangkan di 1970 dengan adanya *Smalltalk*. Meskipun tidak banyak *software developer* menggunakan bahasa *Object Oriented Programing* merka tetap ber evolusi. Pada tahun 1980 an ada kebangkitan perhatian di *Object Oriented* metodologi. Khususnya bahasa *Object Oriented Programing* seperti C++ dan Eifle menjadi populer dengan komputer *mainstream* para programer. *Object Oriented Programing* berlanjut berkembang popularitas di tahun 90 an, terutama terkenal dengan kedatangannya Java dan pengikutnya. Dan pada tahun 2002, bersamaan dengan pelancurannya *.NET Framework*, Microsoft memperkenalkan bahasa *Object Oriented Programing* baru, C# panggilan dari C *sharp*, dan merubah *Visual Basic* sehingga benar benar bahasa *Object Oriented Programing*. (Clark, 2011:2)

2.1.2.2. Karakteristik dari Object Oriented Programing

Sebagian konsep karakteristik objek dasar dan istilah yang umum untuk semua bahasa *Object Oriented Programing* sebagai berikut:

1. *Objects*

   Objek adalah suatu barang seperti printer, dimana metode seperti pengiriman dan pendapatan data atau prosedur untuk bekerja dengan data atau prosedur itu. (Clark, 2011:3)

2. *Abstraction*

   Abstraksi adalah penyaringan data dari sebuah objek darimana objek itu mempunyai banyak data tidak penting yang harus di proses. Dengan abstraksi komunikasi dari objek memiliki yang beda attribut tidak berlebihan. (Clark, 2011:3)

3. *Encapsulation*

   Enkapsulasi adalah proses yang menyembunyikan data agar tidak dapat di lihat, jika ingin melihat datanya harus berinteraksi dengan objek yang mempunyai datanya. Dengan adanya *Encapsulation* data menjadi lebih aman untuk sistem dan terpercayai. (Clark, 2011:4)

4. *Polymorphism*

   Polimorfisme adalah kemampuan dua tau lebih objek untuk menanggapi perintah yang di berikan dalam interpretasi masing masing objek. Seperti menyuruh peliharaan untuk mengomong, tetapi beda peliharaan akan mengeluarkan suara yang beda dengan satu perintah. (Clark, 2011:4)

5. *Inheritance*

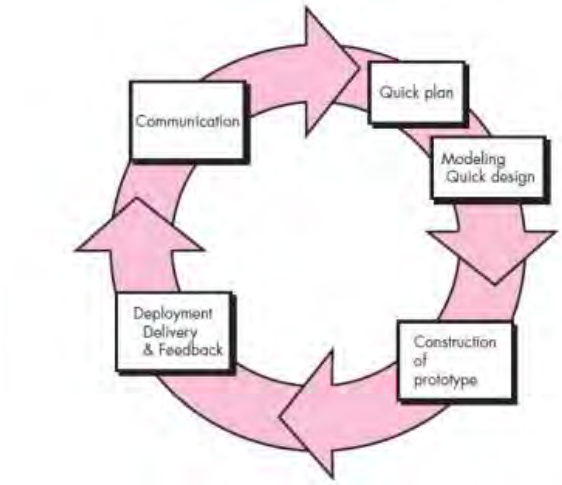
   Sebagian besar objek diklasifikasikan menurut hierarki. Dimana objek dapat di klasifikasikan kedalam tipe karakteristik seperti peliharaan yang mempunyai empat kaki dan berbulu. (Clark, 2011:5)

6. *Aggregation*

   Agregasi adalah ketika satu objek terdiri gabungan objek yang lain untuk berkerja bersama sebagai satu objek yang penuh. Fitur terkuat yang digunakan object orient programing untuk dengan akurat memodelkan dan melaksanakan proses di dalam program. (Clark, 2011:5)

2.1.3. Model Pengembangan *Prototyping*

Prototyping adalah proses model yang dapat dijalani sendiri, p*rototyping* lebih umum digunakan sebagai teknik yang dapat diimplementasikan di dalam model lain. Dalam general objektif, *software* tidak di jelaskan dengan secara detail untuk kebutuhan fungsi dan fitur. *Prototyping* di gunakan sebagai sistem untuk mengenali kebutuhan *software*. Jika *prototyping* yang sudah dibuat, dapat ditambahkan sebagian dari program untuk memungkinkan pembuatan program dengan cepat. (Pressman, 2014:45)



Gambar 2.1
Model *Prototyping*

(Sumber: https://coretanlusuh.files.wordpress.com/2014/05/prototipe.png?w=400&h=318)

Prototyping dapat digunakan sebagai sistem pertama, yang di rekomendasikan untuk di buang. Tetapi dapat di lihat sebagai ideal, walau sebagai *prototyping* di buat untuk di buang, yang lain berevolusi menjadi sistem yang sebenarnya. Implementasi sistem *prototyping* dengan kompromi agar dapat di gunakan secepat mungkin, pengunaan sistem operasi atau bahasa pemograman digunakan karena tersedia dan diketahui, sistem algoritma yang tidak cocok dapat digunakan untuk mendemonstrasikan kemampuan sistem. (Pressman, 2014:46)

2.2. UML

Para developer percaya bahwa UML adalah metodologi, dari akronim “M”. Tetapi itu tidak benar UML yang berarti *Unified Modeling Language*, *language* yang dapat digunakan untuk memberi deskripsi.

Mengenali bahasa bukan artinya dapat menghasilkan hasil yang berguna. Contoh seperti bahasa inggris, orang dapat bicara dengan bahasa inggris tapi bukan artinya mereka dapat menulis poet atau membuat pidato dengan baik. Selain dari sintaksis, ada pengetahuan dan teknik yang dapat digunakan untuk membatu penyair atau pembicara menyusun elemen elemen bahasa berurut dan struktur yang bagus untuk menghasilkan

hasil yang bagus. Tidak semua diagram perlu digunakan untuk pengembangan system. Hanya sebagian infomasi yang berguna saja direkomendasikan untuk proyek. (Wazlawick, 2014:3)

Menurut website https://www.uml-diagrams.org/uml-25-diagrams.html bahwa berikut adalah UML *Structure Diagrams* dan UML *behavior diagrams.*

Tabel 2.1
UML *Structure Diagrams*

Diagram	***Purpose***
Class diagram	Menunjukkan struktur sistem, subsistem atau komponen yang dirancang sebagai kelas dan antarmuka terkait, dengan fitur, kendala, dan hubungannya - asosiasi, generalisasi, dependensi, dll.
Object diagram	Instance level class diagram yang menunjukkan contoh spesifikasi class dan interface (*object*), slot dengan spesifikasi, dan tautan (contoh asosiasi).
Package diagram	Memperlihatkan paket dan hubungan antar paket.
Composite structure diagram	Diagram yang dapat digunakan untuk mengasih lihat *Internal structure of a classifier* dan *A behavior of a collaboration.*
Internal structure diagram	Memperlihatkan struktur internal dari penggolong – dekomposisi penggolong menjadi sifat, bagian, dan hubungannya.
Collaboration use diagram	Menunjukkan objek dalam satu sistem yang bekerja sama satu sama lain untuk menghasilkan beberapa perilaku sistem.
Component diagram	Menampilkan komponen dan dependensi di antara mereka. Diagram jenis ini digunakan untuk *Component-Based Development (CBD)*, untuk menggambarkan sistem dengan *Serviced-Oriented Architecture (SOA).*
Deployment diagram	Menunjukkan arsitektur sistem sebagai penyebab (distribusi) artefak perangkat lunak ke target penyebaran.

Diagram	***Purpose***
	Diagram penyebaran tingkat spesifikasi (disebut juga level level) menunjukkan beberapa gambaran tentang penyebaran artefak ke target penyebaran, tanpa merujuk contoh spesifik artefak atau node. Diagram penerapan tingkat instance menunjukkan penyebaran contoh artefak ke contoh spesifik dari target penerapan. Ini dapat digunakan misalnya untuk menunjukkan perbedaan dalam penyebaran ke lingkungan pengembangan, pementasan atau produksi dengan nama / id dari server atau perangkat pengembangan atau penempatan tertentu.
Network architecture diagram	Deployment diagram bisa digunakan untuk menunjukkan logis atau fisik *network architecture* sistemnya. Jenis diagram penyebaran ini - tidak didefinisikan secara formal dalam UML 2.5 - bisa disebut diagram arsitektur jaringan.
Profile diagram	Diagram UML tambahan yang memungkinkan untuk mendefinisikan stereotip khusus, nilai yang ditandai, dan batasan sebagai *lightweight extension mechanism* untuk standar UML. Profil memungkinkan untuk menyesuaikan metamodel UML untuk berbeda *platforms* (seperti J2EE atau .NET), atau *domains* (seperti pemodelan proses waktu nyata atau bisnis)

Tabel 2.2
UML *Behavior Diagrams*

Diagram	***Purpose***
Use Case Diagram	Menggambarkan serangkaian tindakan (*use cases*) bahwa beberapa sistem atau sistem (*subject*) harus atau dapat melakukan kerja sama dengan satu atau lebih pengguna eksternal dari sistem (*actors*) untuk memberikan hasil yang dapat diamati dan berharga

Diagram	***Purpose***
	kepada para pelaku atau pemangku kepentingan lainnya dari sistem.
Information flow diagram	Menunjukkan pertukaran informasi antara entitas sistem pada beberapa tingkat abstraksi yang tinggi. Arus informasi mungkin berguna untuk menggambarkan sirkulasi informasi melalui suatu sistem dengan merepresentasikan aspek-aspek model yang belum sepenuhnya ditentukan atau dengan rincian yang lebih sedikit.
Activity diagram	Menunjukkan urutan dan kondisi untuk mengkoordinasikan perilaku tingkat yang lebih rendah, daripada penggolong yang memiliki perilaku tersebut. Ini biasanya disebut model *control flow* dan model *object flow*.
State machine diagram	Digunakan untuk memodelkan perilaku diskrit melalui transisi status berhingga. Selain mengekspresikan perilaku bagian dari sistem, *state machine* juga dapat digunakan untuk *usage protocol* bagian dari suatu sistem. Kedua jenis *state machine* disebut sebagai *behavioral state machines* dan *protocol state machines*.
Behavioral state machine diagram	Menunjukkan diskrit *behavior* sebagai bagian dari sistem yang dirancang melalui transisi keadaan yang terbatas.
Protocol state machine diagram	Menunjukan *usage protocol* atau *life cycle* dari dari beberapa penggolong, semisal operasi mana dari classifier dapat dipanggil di setiap negara dari *classifier*, di mana kondisi spesifik, dan memuaskan beberapa opsional *postconditions* setelah classifier bertransisi ke status target.
Interaction diagram	*Iteraction diagram* termasuk dari beberapa jenis diagram seperti *Sequence diagrams, Communication diagrams, Timing diagrams,* dan *Interaction overview diagrams*

Diagram	***Purpose***
Sequence Diagram	Jenis diagram interaksi yang paling umum yang berfokus pada pertukaran pesan antara garis hidup (*objek*).
Communication diagram atau *Collaboration diagram* di UML 1.x	Fokus kepada interaksi atntara *lifelines* di mana internal arsitektur struktur dan *correspond* dengan *message* yang melewati *central*. Urutan *message* diberikan melalui urutan skema *numbering scheme*.
Timing diagram	Menunjukkan interaksi ketika tujuan utama dari *diagram* untuk alasan tentang waktu. *Timing diagram* fokus terhadap pergantian kondisi di dalam dan antara *lifeline* sepanjang sumbu waktu linear.
Interaction overview diagram	Menentukan interaksi melalui varian *activity diagram* dengan cara yang mempromosikan peninjauan luas aliran kontrol. Interaksi *overview diagram* fokus kepada *overview* dari aliran kontrol dimana interaksi atau interaksi yang digunakan. Jalur *lifeline* dan *message* tidak terlihat pada *level overview* ini.

2.3. Definisi *Game*

Menurut Chandler, dalam buku yang berjudul "*Fundamentals of Game Development*", definisi *game* sebagai berikut: "*One might describe a game as a play activity defined by interactive challenges discernible rules, and attainable goals*". (Chandler, 2011:1)

Dari definisi di atas dapat di simpulkan bahwa:

Game adalah sebuah aktifitas bermain yang tergantung dari tantangan interaktif, dan tujuan yang dapat dicapai.

Menurut Roger S, dalam buku yang berjudul "*Level UP! The Guide to Great Video Game Design*", definisi *game* sebagai berikut: "*Playing a game is a voluntary effort to overcome unnecessary obstacles*". (Roger S, 2014:7)

Dari definisi di atas dapat di simpulkan bahwa:

Bermain *game* merupakan upaya sukarela untuk mengatasi kendala yang tidak perlu.

2.3.1. Genre *Games*

Istilah genre digunakan untuk mengkategorikan sesuatu, kategori biasanya menjelaskan buku, film, atau musik. Contohnya untuk musik dapat memiliki genre seperti *rock and roll*, *gospel*, atau *country*. Berikut adalah contoh contoh tipe genre *games* seperti: (Roger S, 2014:16)

1. *Action*

   Game yang membutuhkan koordinasi tangan dan mata untung bermain, genre action juga mempunyai beberapa subgenre.

   a. *Action-adventure*

      Action adventure adalah kombinasi dari genre fitur mengumpuli barang dan penggunaan barang, memecahkan teka-teki, dan cerita janga panjang terkait dengan tujuan. Contoh seperti: *Price of Persia* dan *Tomb Raider*

   b. *Action*-arcade

      Game apa pun yang disajikan dalam gaya arcade awal *game* arcade awal dengan penekanan pada *game* “kedutan”, skor, dan waktu putar yang singkat. Contoh seperti: *Dig Dug*, *Diner Dash*.

   c. Platformer

      Game platform kadang fitur dengan maskot karakter melompat (atau berayun atau bermantul) walaupun di bilang lingkungan *platform*. Menembak dan berkelahi juga mungkin terlibat. Pada satu waktu, platformer adalah subgenre paling populer di dunia *game*. Contoh seperti: Nintendo’s Mario *title* (*Super Mario World*, *Mario* 64, dan *Super Mario Galaxy*).

d. *Stealth*

Game yang beraksi dengan penekanan pada menghindari musuh daripada langsung melawan mereka. Contoh seperti: *Metal Gear* series dan *Thief*: *The Dark Project*.

e. *Fighting*

Game dimana dua tau lebih musuh di dalam arena. *Game* berkelahi berbeda dari *game* aksi utamanya dari kedalaman kontrol mereka. Contoh seperti: *Street Fighter* series dan *Mortal Comat* series.

f. *Beat em up*/*hack and slash*

Game yang mengasih musuh dalam *wave* dengan menambah kesusah. Contoh seperti: *Double Dragon*, *Castle Crashers*.

2. *Shooter*

Utamanya *game shooter* adalah *game* yang menembak proyektil, *game* shooter banyaknya berevolusi di bidang kamera.

a. *First Person Shooter*

Game tembak tembakan yang di mainkan dalam perspektif penembak. Contoh seperti *Quake*, *Team Fortress 2*.

b. Shoot em up

Game yang menembak *arcade style*, menembak dengan jumlah yang banyak selagi menghindari bahaya. Contoh seperti: *Space Invader*, *the Contra series*.

c. *Third Person Shooter*

Game tembak tembakan yang dimana kameranya di letakan di belakang karakter, memberikan lebih besar jarak pandang. Contoh seperti: *Star Wars Battlefront*, dan *Grand Theft Auto series*.

3. *Adventure*

Game adventure dimana permainan akan fokus pada memecahkan tekateki, dan mengumpuli barang.

a. *Graphical Adventure*

Game yang menarik di bagian grafis dan suasana *game*, menggerakan kursor untuk berjalan jalan dan mencari petunjuk. Contoh seperti: *Myst*, *Monkey Island*, dan the *Sams and Max series*.

b. *Role Playing Game*

Sub genre ini di ambil dari *game* papan seperti *Dungeon and Dragons* dimana karakter digerakan melewati peta, masing masing karakter memiliki kelas dan *abilities* yang melewati kombat, explorasi, dan mencari harta karun. Contoh seperti: *Star Wars: Knights of the Old Republic* dan the *Mass Effect series*.

c. *Massively Multiplayer Online Role Playing Game*

Game RPG yang mendukung ratusan pemain di satu peta. *Game* MMORPG dikenal dari *player versus player*, mencari harta ber ulang-ulang "*grinding*", dan perang group "*raids*". Contoh seperti: *World of Warcraft*, *DC Universe Online*.

d. *Survival/Horror*

Game dimana harus bertahan hidup di skenario horor dengan bahan baku yang terbatas, seperti amunisi yang sedikit. Contoh seperti: *Resident Evil series*, the *Silent Hill series*.

4. *Construction/management*

Genre yang dimana pemain harus membagnung dan memperluas bangunan dengan lokasi dan bahan baku yang terbatas. Contoh seperti: *SimsCity* dan *Zoo Tycoon*.

5. *Life Simulation*

Game yang mirip dengan *game* yang mengatur atur relasi para AI/NPC di dalam *game*. Contoh seperti: *The Sims* and the *Princess Maker*.

a. *Pet Simulation*

Simulasi dimana pemain merasakan memelihara peliharaan dengan menjaga, mengasih makan, dan memperkuat relasi pemain dengan peliharaan. Contoh seperti: *Wolrd of Zoo*, dan *Tamagotchi*.

6. *Music/rhythm*

   Game yang membikin pemain untuk mengikuti irama musik agar mendapatkan point score. Contoh seperti: *Voez*, dan *Osu*.

7. *Party*

   Game Party biasanya di buat untuk dimainkan oleh banyak player, dimana mereka akan saling berlawanan untuk dempatakan skor. Contoh seperti: *Mario Party*, dan *Buzz*.

8. *Puzzle*

   Game Puzzle berbasis dari *game* logistik dan mencocokan pola, *game* bisa sangat lama dan melatih *hand*/*eye coordination*. Contoh seperti: *Incredible Machine* atau *Tetris*.

9. *Sports*

   Game yang berbasis dari kompetisi atletis tradisional atau extrim. Contoh seperti: the *Madden series*, the *Tony Hawk series*.

   a. *Sports Management*

      Daripada bermain secara langsung, pemain memerintah sekelompok karakter atau team. Contoh seperti: *FIFA Manager series*, dan *NFL Head Coach series*.

10. *Strategy*

    Game yang di tarik dari catur, memikir, dan merencanakan langkay yang akan di ambil, adalah keunggulan dari *game* strategi. Mereka bisa di adakan pada jaman dulukala atau dunia masadepan.

    a. *Real Time Strategy*

       Mirip seperti *turn based game* tapi memiliki waktu yang *real time*, yang fokusnya di utaman pada expansi, explorasi, expansi, dan exterminasi. Contoh seperti: *Command and Conquer series*, dan *Warcraft 1-3 series*.

    b. *Turn-based*

       Turn base adalah *game* yang lebih pelan, memberikan pemain untuk berfikir, memberikan kesempatan untuk strategi untuk di gunakan. Contoh seperti: *X-Corn series*, dan *Advance Wars series*.

c. *Tower Defense*

Game sub genre yang baru baru keluar di PC dan *hand phone* dimana pemain menempatkan *tower* yang secara automatis menembak musuh dengan automatis. Contoh seperti: *Defense Grid: The Awakening*, dan *Lock's Quest*.

11. *Vehicle simulation*

Pemain dapat meresakan simulasi kendaraan, dengan perasaan asli.

a. *Driving*

Pemain dapat memperkuat kendaraan, mau dari motor sampai *hovercraft*. Contoh seperti: *Grand Turismo series*, dan *NASCAR Racing series*.

b. *Flying*

Pemain dapat mengendarakan pesawat terbang, dan juga dapat mengendarakan pesawat jet yang sedang bertempur. Contoh seperti: *Starfox*, dan *X-Wing*/*TIE Fighter series*.

2.3.2. Sejarah Pendek *Game Real Time Strategy*

Pada tahun 1983 *game* rts pertama Stonkers di buat untuk sistem 48K ZX Spectrum. Dimana pemain dan komputer diberikan pasukan yang sama. Pasukannya dapat bergerak di map dan menyerang musuh. Dan inovasi paling besar adalah sistem "Rock, Paper, Scissors", kepada konter pasukan. Dan *game* menampilkan basis sistem sumberdaya dimana, pasukan harus disediakan makanan untuk bertahan hidup. (Riskas, 2017:10)

Kejadian yang paling pentik untuk *game* genre *real time strategy* terjadi pada tahun 1989 dari sistem *Sega Mega*, dengan *game* bernama Herzog Zwei. *Game* pertama yang menyerupai *game real time strategy* masa depan. Elemen elemen yang utama seperti pergerakannya, mengatur bahan baku, melatih pasukan, dan *multiple split screen*. (Riskas, 2017:11)

Game lain yang membuat basis untuk *game real time strategy* keluar pada tahun 1992 dengan *game* bernama *Dune II: The Building of a Dynasty* di buat oleh perusahaan *Westwood Studios*, game menyediakan semua elemen elemen yang utama dari *game* modern *real time strategy*, dan di tambah dengan *on-map resource* dan pergerakan

pasukan secara *real time*. Setelah 3 tahun kesuksesan *Dune II, Westwood Studio* mengeluarkan *game* selanjutnya yang bernama *Command and Conquer*. Dengan fitur fitur yang mereka miliki *Command and Conquer* melebihkan dengan UI yang dapat di akses dengan gampang, dan lebih gampang di mengerti. (Riskas, 2017:12)

Pada tahun yang sama, pada tahun 1995 *Blizzard Entertainpemt* mengeluarkan *game* lanjutan mereka, *Warcraft II: Tides of Darkness*. Dari elemen elemen yang digunakan oleh mereka adalah *fog of war*, dan *game* mempunyai *elemen naval units* yang belom di lihat dari *game* lainnya. (Riskas, 2017:12)

Blizzard entertainment tahu bahwa kesuksaesan *game real time strategy* adalah koneksi dari *game real time strategy* yang dulu, karena itu mereka mengeluarkan lanjutan dari *game* masing masing dengan sesuatu yang unik pada masing masing *game*, *game* bernama *Warcraft III: Region of Chaos* dan *Starcraft II*

2.3.3. Definisi *Game Real Time Strategy*

Cara mengontrol pasukan dengan *mouse* dan *keyboard*, dengan *mouse* gerakan yang paling penting, karena alat ini satu satunya yang dapat menyuruh pasukan untuk bergerak. Dan sebagian dari perintah menggunakan jalan pintas *keyboard*. Kebanyakan *game real time strategy* keluar di PC, dan sebagian *game* juga di pindahkan ke *consoles*. Banyaknya *game real time strategy* menyediakan mode *single player* dan *skirmish* mode, melawan AI atau *multiplayer mode*. (Riskas, 2017:15)

Biasanya strategi mengacuh pada kombinasi dari berbagai cara yang digunakan, untuk mencapai kemenangan. Strategi adalah keputusan jangan panjang yang tidak dinilai spontan tetapi dalam jangka waktu. Aspek dasar dari strategi termasuk ekonomi dan diplomasi sebanyak kekuatan militer.

Sisi taktik militer, menurut definisi, ilmu dan seni mengatur kekuatan militer, dan teknik untuk menggabungkan dan menggunakan senjata dan unit militer untuk terlibat dan mengalahkan musuh dalam pertempuran. Dalam sisi lain taktik menyerupai keputusan jangka pendek. Mereka tidak diragukan lagi dapat berguna ketika dalam pertempuran

seperti dalam memerintah pasukan untuk bergerak, menghundari atau menyerang, atau bahkan memutuskan formulasi yang paling menguntungkan apa yang unit harus ambil.

Real time strategy *game* didasarkan pada keseimbangan antara keduanya. Secara teori, strategi atau manajemen makro secara default lebih penting. Pemenenang yang keluar dari sesi biasanya yang bisa membaca permainan lebih akurat. (Riskas, 2017:18)

2.3.4. Ciri *Game Real Time Strategy*

Cara utama menggerakan *game real time strategy* menggunakan *mouse* dan *keyboard*. *Mouse* mempunya kontrol yang paling penting, karena digunakan untuk memilih *units* dan memberi perintah. Banyak perintah dapat dilakukan juga menggunakan *keyboard shortcut*. *Game real time strategy* terjadi di sebuah peta, dan tersedia *resources*. *Minimap* adalah sebuah peta yang biasanya teletak di bawah layer, untuk membantu pemain untuk melacak aksi dari musuh maupun teman. Explorasi adalah tema umum dari *game real time strategy* dimana setiap *game* di mulai pemain hanya dapat melihat sedikit bagian dari peta. (Riskas, 2017:15)

2.4. Unity & NavMesh

Menurut website https://unity3d.com/unity bahwa unity digunakan untuk membuat dunia *game*. Alat real-time dan fleksibel menawarkan kemungkinan luar biasa untuk pengembang *game*, dan pembuat konten di berbagai industri dan aplikasi. Contoh contoh fitur unity seperti:

1. *Unity Editor*

   Editor Kesatuan menampilkan beberapa alat yang memungkinkan pengeditan dan iterasi yang cepat dalam siklus pengembangan Anda, termasuk mode Putar untuk pratinjau cepat pekerjaan secara waktu nyata:

   a. Semua di satu editor

      Tersedia di Windows dan Mac, mencakup berbagai alat yang ramah seniman untuk merancang pengalaman yang menarik dan dunia *game*, serta serangkaian alat

pengembang yang kuat untuk mengimplementasikan logika permainan dan permainan berkinerja tinggi.

b. 2D dan 3D

Unity mendukung pengembangan 2D dan 3D dengan fitur dan fungsionalitas untuk kebutuhan spesifik masing masing.

c. AI *pathfinding tools*

Unity mengandung sistem navigasi yang memungkinkan pembuatan npc yang dapat dengan cerdas bergerak di dunia *game*. Sistem ini menggunakan jaringan navigasi yang dapat secara automatis dari *geometry*, atau objek dinamis, untuk mengubah navigasi karakter pada saat *runtime*.

d. User Iterface

Sistem internal UI Unity meungkinkan pembuatan antarmuka dengan cepat dat intuitif.

e. Physics Engines

Manfaatkan dukungan Box2D dan NVIDIA PhysX untuk permainan yang sangan realistis dan berkinerja tinggi.

f. Custom Tools

Andan dapat memperluas *Editor* dengan alat apapun yang dibutuhkan untuk mencocokan alur kerja tim. Buat dan tambahkan ekstensi khusus atau temukan apa yang di butuhkan dari Toko Aset Unity, yang menampilkan ribuan sumber daya, alat, dan ekstensi untuk proyek.

2. Unity *Platforms*

Unity menyediakan lebih dari 25 platform, dari mobile, desktop, consle, TV, VR, AR, dan Web. Dengan berbagai macam platform unity dapat menyebarkan *game* yang di buat ke semua pemakai.

3. *Top Performance*

Optimasi kreasi interaktif dengan mesin berkinerja tinggi yang terus meningkat.

a. Alat profil canggih menawarkan wawasan, seperti menentukan apakah *game* CPU atau GPU-*bound*, dan bagaimana mengoptimalkan kinerja *rendering* dan *gameplay* untuk pengalaman yang halus.
b. Kinerja C++ di seluruh platform dengan skripsi IL2CPP *back-end* yang selalu dikembangkan oleh Unity.
c. *Scripting runtime* Mono/.NET 4,6/C#6 (pencobaan di 2017.1)

4. *Grafik Rendering*

Hidupkan *game* dengan sinar matahari dari siang hari atau cahaya lampu neon pada malam hari.

a. *Real-time rendering engine*

Buat grafik yang luar biasa dengan *real time global* iluminasi dan *physically based rendering*

b. *Native Graphics* APIs

Unity mendukung multiplatform, tetapi tetap dekat dengan API grafis tingkat rendah dari setiap *platform*, memungkinkan untuk memanfaatkan peningkatan GPU dan *hardware* terbaru, seperti Vulkan, uIS Metal, DirectX12, nVidia VRWorks, atau AMD LiquidVR.

5. *Artist and Designer Tools*

Unity Editor adalah pusat keratif untuk para seniman, desainer, pengembang, dan anggota tim lainnya. Tremasuk alat desain adegan 2D dan 3D, menceritakan dan sinematik, pencahayaan, sistem audio, alat manajemen *Sprite*, efek partikel dan sistem animasi *dopesheet* yang kuat.

a. *Storytelling*

Alat *Timeline* menyediakan para artis kekuatan untuk membuat *stunning cinematic* & *gameplay sequence*.

b. *Cinematic content*

Dengan rangkaian kamera yang cerdas dan dinamis, pengontrolan bidikan seperti sutradara filem dari dalam editor.

c. *Color grading and effects*

Pembuatan *game* yang terlihan profesional dan dengan penuhnya menggunakan *Post Processing FX*.

d. *Animation*

Menggunakan *Timelinei*, *Anima2D*, *Particles*, integrasi yang erat dengan Maya dan tools yang lainnya untuk menganimasi.

e. *Level Design and Worldbuilding*

Dengan ProBuilder, dengan cepat mendesain, membuat prototipe, dengan menguji coba, lalu memadukan tekstur dan warna, memahat *mesh* dan menyebarkan objek dengan *Polybrush* (*beta*).

f. *Round-tripping*

Detail dan poles model 3D secara langsung dengan integrasi tanpa batas alat ciptaan konten digital (DCC) seperti Maya.

g. *Lighting*

Dapatkan pendapat yang instant dengan *Progressive Lightmapper*, dan optimalkan adegan dengan mode *Mixel Lighting* untuk hasil yang paling baik.

6. Unity *Asset Store*

Pada saat membuat *game* jika ada masalah, Unity *Asset Store* sudah menyediakan banyak solusi.

a. Off-the-shelf konten untuk menambahkan proyek Unity dan membuat pengembangan lebih cepat dan gampang.
b. Katalog besar untuk pemakai berbayar atau geratis.
c. Persediaan unity yang di sediakan: *Art*, *Models*, *Scripts*, *Productivity Tools*, dan lain lain.

Menurut website https://docs.unity3d.com/Manual/nav-NavigationSystem.html *NavMesh* dikatakan bahwa *NavMesh* adalah sistem navigasi yang memungkinkan untuk membuat karakter yang dapat navigasi dunia *game*. Ini memberi karakter kemampuan

untuk memahami bahwa mereka perlu naik tangga untuk mencapai lantai dua, atau melompat untuk melewati objek.

1. *NavMesh* (kependekan dari *Navigation Mesh*) adalah struktur data yang menggambarkan permukaan yang dapat di jalani di dunia *game* dan memungkinkan untuk menemukan jalur dari satu lokasi *walkable* ke yang lain di dunia *game*. Struktur data dibangun, atau dipanggang, secara otomatis dari *geometri* level peta.
2. Komponen NavMesh Agent membantu membuat karakter yang saling menghindari satu sama lain saat bergerak menuju tujuannya.
3. Komponen *Off-Mesh Link* memungkinkan untuk memasukkan pintasan navigasi yang tidak dapat direpresentasikan menggunakan permukaan *walkable*. Misalnya, melompati pagar, atau membuka pintu sebelum berjalan melaluinya, semua dapat digambarkan sebagai tautan *Off-mesh*.
4. Komponen NavMesh Halangan memungkinkan untuk menggambarkan hambatan yang bergerak agar agen dapat menghindari saat menavigasi dunia. Sebuah tong atau peti yang dikendalikan oleh sistem fisika adalah contoh yang baik dari hambatan.

Algoritma yang sering dipakai untuk mencari jalur adalah A*, yang digunakan oleh unity. Notasi yang di pakai oleh Algoritma A* adalah sebagai berikut:

f(n) = g(n) + h(n)

f(n) = biasanya esetemasi terendah

g(n) = biaya dari *node* awal ke *node* n

h(n) = perkiraan biaya dari *node* n ke *node* akhir

Dalam penerapannya, Algoritma A* memiliki beberapa terminologi dasar diantaranyastarting point, simpul (nodes), A, open list, closed list, harga (cost), halangan (unwalkable).

1. *Starting point* adalah sebuah terminologi untuk posisi awal sebuah benda.
2. A adalah simpul yang sedang dijalankan dalam algortima pencarian jalan terpendek.

3. Simpul adalah petak-petak kecil sebagai representasi dari area pathfinding bentuknya dapat berupa persegi lingkaran, maupun segitiga
4. *Open list* adalah tempat menyimpan data simpul yang mungkin diakses dari *starting point* maupun simpul yang sedang dijalankan.
5. *Closed list* adalah tempat menyimpan data simpul sebelum A yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.
6. Harga adalah nilai yang diperoleh dari penjumlahan, jumlah nilai tiap simpul dalam jalur terpendek dari *starting point* ke A, dan jumlah nilai perkiraan dari sebuah simpul ke simpul tujuan.
7. Simpul tujuan yaitu simpul yang di tuju.
8. Halangan adalah sebuah atribut yang menyatakan bahwa sebuah simpul tidak dapat dilalui oleh A.

2.5. Bahasa Pemograman C#

Menurut Svetlin Nakov dan Veselin Kolev, dalam buku yang berjudul "*Fundamentals of Computer Programming with* C#", definisi *object oriented programing* adalah sebagai berikut:

"C# *is a modern object-oriented, general-purpose programming language, created and developed by Microsoft together with the .NET platform. There is highly diverse software developed with C# and on the .NET platform: office application, web application, websites, desktop applications, mobile applications, games and many others*." (Nakov & Kolev, 2013:22)

Dari definisi di atas dapat disimpulkan bahwa:

C# adalah bahasa pemrograman berorientasi objek, bertujuan umum yang moderen, dibuat dan dikembangkan oleh Microsoft bersama dengan platform .NET. Dan banyaknya perangkat lunak yang dikembangkan oleh mereka seperti: aplikasi kantor, aplikasi web, situs web, aplikasi *desktop*, aplikasi seluler, *game*, dan banyak lainnya.

C# berbahasa pemograman yang moderen, tersebar luas, dan banyak di pakai oleh programmer di sekelilig dunia. Selain itu C# sangat simple dan gampang di pelajari dari pada C dan C++. Wajar untuk memulai dengan bahasa yang cocok untuk pemula,

sementara masih banyak digunakan di industri besar, membikin C# salah satu bahasa pemograman yang paling terkenal saat ini. (Nakov & Kolev, 2013:23)

Meskipun banyak diskusi antara menggunakan C# atau Java, secara umum di akui bahwa Java adalah pesaing yang paling serius. Tetapi tidak perlu dipersoalkan yang mana lebih bagus karena C# lebih kuat, lebih kaya, dan lebih direkayasa. (Nakov & Kolev, 2013:23)

Keunggulan dari C#, karena mereka menggunakan bahasa pemograman *object oriented*. Bahasa pemograman *object oriented* di pikirkan sebagai bahasa yang dapat berkerja dengan objek di dunia asli, contoh seperti murid, sekolah, buku. Barang mempunyai property seperti nama, warna, dapat melakukan aksi seperti bergerak, berbicara, dan lain lain. (Nakov & Kolev, 2013:25)

2.6. Biologi

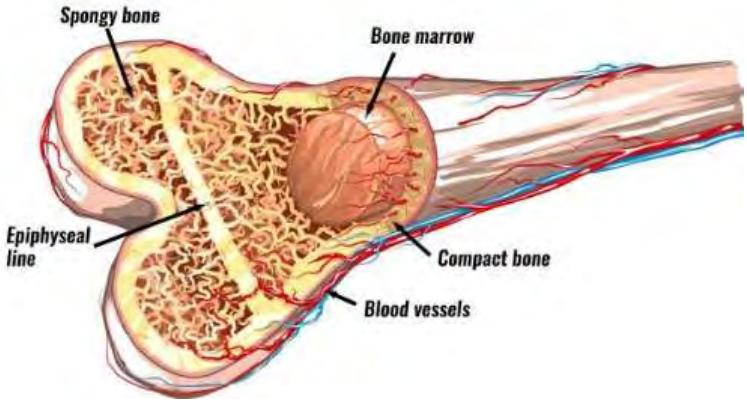
2.6.1. *Immune System*

1. *Physical Barriers*

   Pertahanan tubuh pertama dari para penyerang adalah penahan fisik seperti kulit, untuk membikin susah virus, parasite, bakteria untuk menyerang tubuh. Kulit terdiri dari sekitar 2 meter kulit yang mengelilingi tubuh. (Sompayrac, 2015:1)

2. *Bone Marrow*

   Menurut website https://www.ucsfbenioffchildrens.org/education/what_is_bone_marrow/ bahwa *bone marrow* adalah substansi seperti *sponge* teletak di tengah tulang. Memproduksi *stem cells* dan substansi lain, yang menghasilkan *cell* darah masing masing dari *cell* yang di buat oleh *bone marrow* mempunyai kerjaan yang penting.

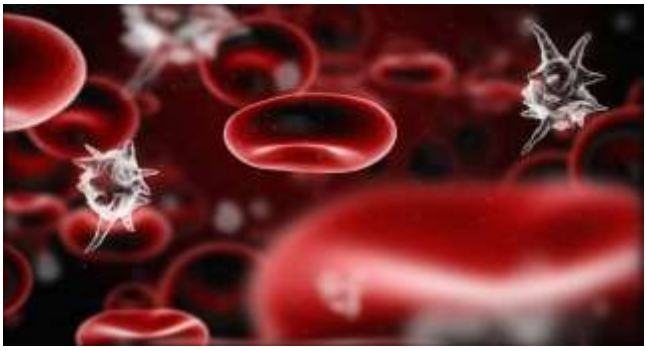


Gambar 2.2
Bone Marrow

(Sumber: https://www.reachchildcancer.org.nz/images/blog/5cbffcc3b1a4abe60cb79b2f9afee889__3854/fit1200x1200.jpg)

a. *Platelets* (*Trombocytes*)

Menurut website http://www.hematology.org/Patients/Basics/ bahwa *Platelets* tidak seperti sel darah merah dan putih, trombosit sebenarnya bukan sel melainkan fragmen sel kecil. Trombosit membantu proses pembekuan darah (atau koagulasi) dengan berkumpul di tempat cedera, menempel pada lapisan pembuluh darah yang terluka, dan membentuk platform di mana pembekuan darah dapat terjadi. Ini menghasilkan pembentukan gumpalan fibrin, yang menutupi luka dan mencegah darah bocor keluar. Fibrin juga membentuk perancah awal di mana bentuk jaringan baru, sehingga mempromosikan penyembuhan. Jumlah trombosit yang lebih tinggi dari normal dapat menyebabkan pembekuan yang tidak perlu, yang dapat menyebabkan stroke dan serangan jantung; Namun, berkat kemajuan yang dibuat dalam terapi antiplatelet, ada perawatan yang tersedia untuk membantu mencegah kejadian yang berpotensi fatal ini. Sebaliknya, jumlah yang lebih rendah dari normal dapat menyebabkan perdarahan yang luas.



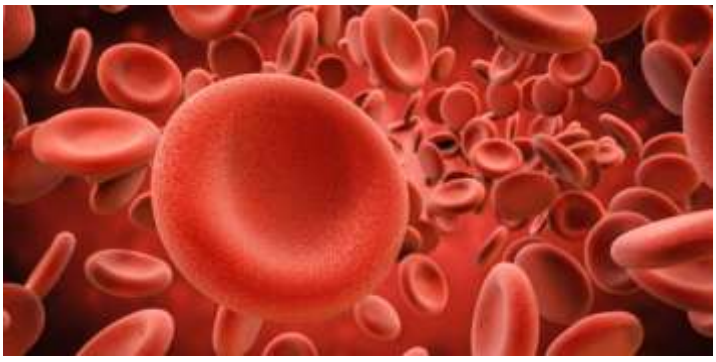
Gambar 2.3
Trombocytes

(Sumber: https://www.thrombocyte.com/wp-content/uploads/2015/07/what-are-platelets.jpg)

b. *Red Blood Cell*

Menurut website http://www.hematology.org/Patients/Basics/ bahwa darah adalah cairan tubuh khusus yang memiliki empat komponen utama: plasma, sel darah merah, sel darah putih, dan trombosit. Darah memiliki banyak fungsi berbeda, termasuk:

1) Membawa oksigen dan nutrisi ke paru-paru dan jaringan.
2) Membentuk gumpalan darah untuk mencegah kehilangan darah berlebih.
3) Membawa sel dan antibodi yang melawan infeksi.
4) Membawa produk limbah ke ginjal dan liver, yang menyaring dan membersihkan darah.
5) Mengatur suhu tubuh.

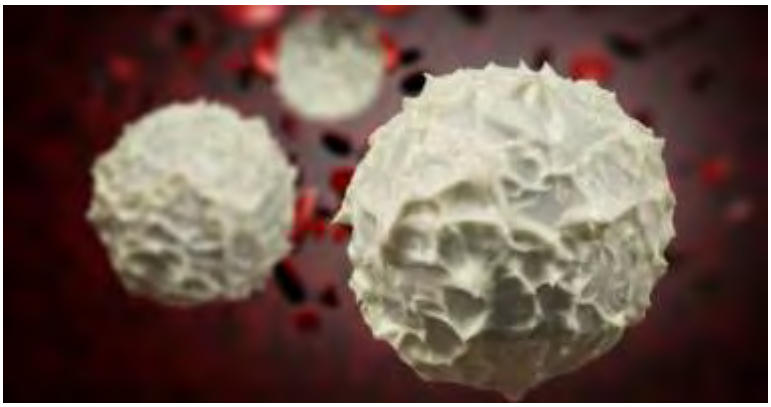


Gambar 2.4
Red Blood Cell

(Sumber: https://labiotech.eu/wp-content/uploads/2018/04/EryDel-Red-Blood-Cells.jpg)

c. *White Blood Cell*

Menurut website http://www.hematology.org/Patients/Basics/ bahwa sel darah putih melindungi tubuh dari infeksi. Jumlahnya jauh lebih sedikit daripada sel darah merah, terhitung sekitar 1 persen dari darah Anda. Jenis sel darah putih yang paling umum adalah neutrofil, yang merupakan sel "respon langsung" dan menyumbang 55 hingga 70 persen dari jumlah sel darah putih total. Setiap neutrofil hidup kurang dari satu hari, jadi *bone marrow* harus secara konstan membuat neutrofil baru untuk menjaga perlindungan terhadap infeksi. Transfusi *neutrofil* umumnya tidak efektif karena mereka tidak berada di dalam tubuh untuk waktu yang lama.

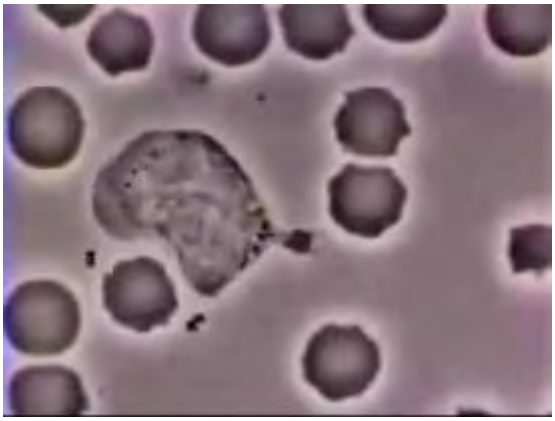


Gambar 2.5
White Blood Cell

(Sumber: https://www.thrombocyte.com/wp-content/uploads/2015/07/what-are-platelets.jpg)

d. *Macrophage*

Ketika penyerang melewati kulit kita mereka akan menemukan *Macrophage*, *Macrophage* tidak hanya diam saja, bahkan mereka dapat mendeteksi penyerang dengan menggunakan *antennae* mereka yang special untuk mengenal penyerang. *Macrophage* berasal dari kata "*Macro*" yang berarti cell besar, dan "*Phage*" berasal dari *Greek* yang berarti untuk makan. Mereka juga dapat memanggil cell lain untuk ikut melawan para *invader*. (Sompayrac, 2015:2)



Gambar 2.6
Macrophage

(Sumber: https://i.ytimg.com/vi/5M1nUjxWAQI/hqdefault.jpg)

e. Eosinophils

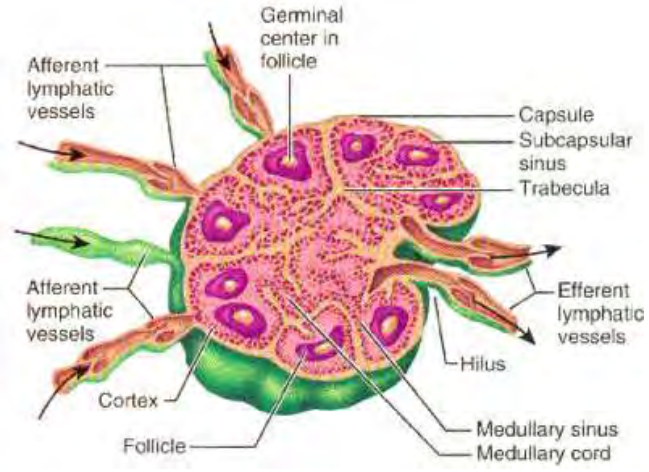
Menurut website https://www.webmd.com/ bahwa *eosinophils* adalah semacam *white blood cell* yang menolong melawan penyakit. Perkerjaan *Eosinophils* tidak terlalu jelas, tetapi biasanya terkait dengan penyakit alergi dan sebagian infeksi.

Gambar 2.7
Eosinophils

(Sumber: https://www.news-medical.net/image.axd?picture=2019%2F1%2FBy_somersault1824.jpg)

3. *Lymph Node*

Lymph node adalah tempat di mana Virgin T Cell, dan Virgin B Cell menunggu sampai di aktifkan. Ketika *virgin* T dan B cell berkembang biak dengan cepat sebagian menjadi *Memory* T atau B Cell dimana mereka akan tinggal di dalam tubuh, jika virus yang menyerang balik lagi, mereka sudah siap untuk melawannya. (Sompayrac, 2015:10)



Gambar 2.8
Lymph Node

(Sumber: https://cllsociety.org/wp-content/uploads/2015/12/diagram1.jpg)

a. *Adaptive Immune System*

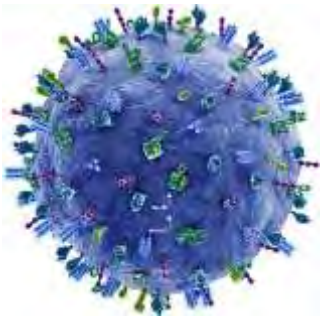
Adaptive Immune System adalah sistem pertahanan yang dapat beradaptasi untuk menjaga kita dari hampir semua serangan. (Sompayrac, 2015:4)

1) *Antibody*

Antibody digunakan untuk menempel ke musuh agar dapat di kalahkan lebih gampang. (Sompayrac, 2015:4)

2) *B Cell*

Yang memproduksi *antibody* adalah B Cell, dimana B Cell akan menggunakan informasi *genetic* mereka untuk membikin *antibody*. B Cell lahir di Bone Marrow. (Sompayrac, 2015:4)



Gambar 2.9
B Cell

(Sumber: https://www.plengegen.com/wp-content/uploads/B-cell.jpg)

3) *T Cell*

T Cell juga termasuk dalam *Adaptive Immune System,* mereka lahir dari *thymus* karena itu mereka di panggil "T" *cell*, *T Cell* sendiri mempunyai 3 tipe: (Sompayrac, 2015:7)

a) *Killer T Cell*

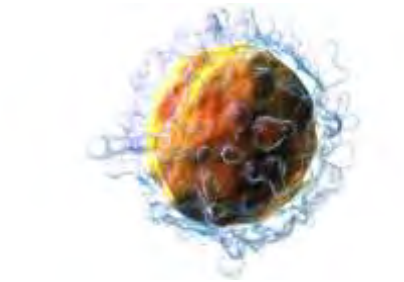
Killer T *Cell* adalah senjata kuat yang dapat menghancurkan cell yang terinfeksi oleh virus.

b) *Helper T Cell*

Helper T Cell adalah cell yang membatu *immune system* yang lain, dengan mengirimkan pesan kimia yang menyemangatkan cell dari *immune system*.

c) *Virgin T Cell*

Virgin T *Cell* adalah cell yang tidak berapa berguna sampai mereka di aktifkan, ketika di aktifkan t *cell* akan berkembang biak dengan cepat.

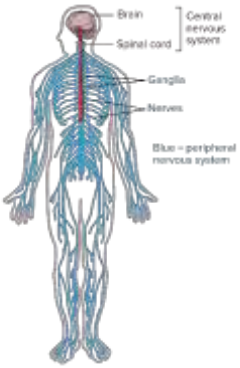


Gambar 2.10
T Cell

(Sumber: https://cdn.the-scientist.com/assets/articleNo/30036/iImg/774/uploads/t%2520cell%2520full.jpg)

4. *Nervous System*

Menurut website https://www.livescience.com/22665-nervous-system.html bahwa *nervous System* adalah koleksi dari kompleks *nerves* dan *neurons* yang mengirim signal antara bagiana tubuh. Dengan singkatnya *Nervous System* adalah alur kabel listrik tubuh manusia.

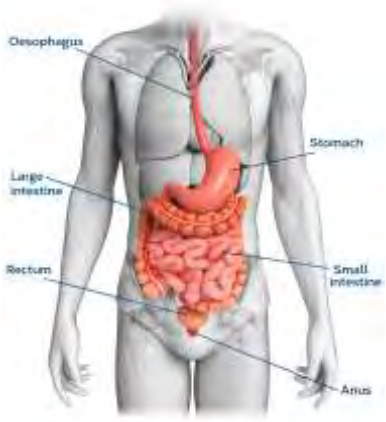


Gambar 2.11
Nervous System

(Sumber: https://cdn.kastatic.org/ka-perseus-images/f7fa1bdddc896c2c635a1929c6578b745a6030c2.png)

5. *Intestine*

   Menurut website https://www.webmd.com/ bahwa *intestine* adalah tabung panjang dari lambung sampai *anus*. Kebanyakan dari nutiri dan air di serap di *intestine*. Yang termasuk *intestine* mulai dari *intestine* kecil, *intestine* besar, dan *rectum.*



Gambar 2.12
Intestine

(Sumber: https://enterofytol.be/wp-content/uploads/2016/07/schema-systeme-digestif-EN.gif)

6. *Artery*

   Menurut website https://www.webmd.com/ bahwa *Artery* adalah saluran darah untuk mengirim darah beroksigen dari hati ke seluruh bagian tubuh.

Gambar 2.13
Artery

(Sumber: https://cdn.britannica.com/49/115249-050-0DFBBCD3/Human-circulatory-system.jpg)

7. *Fat*

Menurut website https://medlineplus.gov/ency/patientinstructions/000104.htm bahwa *Fat* yang di makan memberi energi kepada tubuh untuk berkerja dengan benar. Selagi berolahraga, tubuh menggunakan kalori dari karbohidrat dari makanan.

Gambar 2.14
Fat

(Sumber: https://cdn.mos.cms.futurecdn.net/qZ3MRVJsvwUZpWH9EX6Bhc-1200-80.jpg)

2.6.2. *Attacking Microbes*

Bagi kebanyakan orang mengira virus dan bakteri sangat mirip. Tapi virus dan bakteri tidak mirip sama sekali, memahami bagaimana mikroorganisme yang berbeda bertahan di tempat pertama adalah fundemental terhadap kemampuan kita untuk mengobati penyakit yang ditimbulkannya. (Wiles, 2017:1)

1. *Bacteria*

   Bakteri berukuran mikroskopis, mulai dari 0,5 hingga 5 mikron. Untuk perbandingan sel manusia berkisar 7 hingga 120 mikron dengan diameter. Bakteri terdiri dari zat lengket yang disebut *sitoplasma*. Di sinilah semua proses biokimia yang berlangsung, diatur oleh set tertentu dari instruksi genetik yang disandikan oleh DNA mereka**.** Bakteria berkembang biak dengan membelah diri menjadi dua, tapi membutuhkan 20 sampai 30 menit, mengubah satu bakteria menjadi jutaan dalam 10 jam. (Wiles, 2017:1)

Gambar 2.15
Streptococcus Pneumoniae

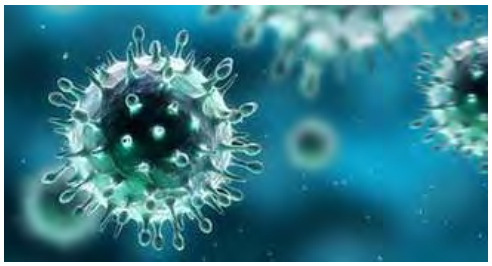
(Sumber: https://www.cdc.gov/pneumococcal/images/s_pneumoniae.jpg)

Gambar 2.16
Helicobacter Pylori

(https://badgut.org/wp-content/uploads/Image-Content-hpylori-1.png)

2. *Virus*

Dibandingkan bakteri, virus jauh lebih sederhana dan jauh, jauh lebih kecil, berukuran antara 0,02 dan 0,2 mikron. Virus terdiri dari kulit terluar yang disebut kapsid yang mengelilingi materi genetik mereka, yang bisa berupa DNA atau RNA. Sebenarnya, virus tidak hidup. Setelah seseorang diambil oleh sel inangnya ia membajak sel dan mendapatkannya untuk membaca viros materi genetik maka sel akan mulai mereplikasi virus. (Wiles, 2017:2)



Gambar 2.17
Influenza

(Sumber: http://www.shl.uiowa.edu/dcd/influenza/influenza.jpg)

3. *Parasite*

Parasit didefinisikan sebagai organisme yang hidup di atau di organisme inang dan mendapatkan makanan mereka dari atau dengan mengorbankan tuan rumah mereka. Seperti bakteria, mereka terdiri dari sitoplasma yang dikelilingi oleh membran sel, meskipun beberapa juga tertutup di permukaan luar yang disebut pelikel. Tidak seperti bakteria, tetapi mirip dengan manusia dan hewan. (Wiles, 2017:3)

Gambar 2.18
Anisakis

(Sumber: https://www.nauticalnewstoday.com/wp-content/uploads/2016/08/anisakis-3-816x600.jpg)

BAB III

ANALISIS DAN PERANCANGAN PERANGKAT LUNAK

3.1. Gambaran Umum Perangkat Lunak

Perangkat lunak yang nantinya akan dikembangkan sebagai *Game Cells* yang dijalankan di komputer. Dimana *game* akan mengajarkan tentang sistem pertahanan tubuh, dengan perspektif yang berbeda. Karena *game* akan menggambarkan kesusahan menjaga tubuh agar tetap sehat, setiap detik tubuh manusia di serang oleh musuh yang tidak terlihat.

Ketika pemain mulai bermain pemain di berikan satu bangunan yang bernama *Nerve Center* yang harus di pertahankan, jika *Nerve Center* hancur *game* akan berakhir tanpa *score game* tersebut di *save*. Untuk aksi yang pemain dapat lakukan di dalam *Game Cells* sebagai berikut:

1. *Select Units*, pemain dapat memilih unit untuk digerakan, pertama pemain harus menekan mouse kiri untuk memilih unit tersebut. Pemain juga dapat menekan tahan mouse kiri dan gerakan mouse untuk memilih semua unit yang di dalam kotak pilihan.
2. Pergerakan Units, pemain harus sudah memilih unit atau units untuk menggerakan unit tersebut. Untuk pergerakan *units* yang dapat di lakukan sebagai berikut:
    a. *Attack*, untuk pergerakan *attack* pemain harus menekan mouse kanan di musuh atau menekan tombol *attack* di bawah ui terus menekan mouse kiri untuk menyuruh units untuk menyerang target.
    b. *Move*, untuk pergerakan *move* pemain harus menekan mouse kanan di tanah atau menekan tombol *move* di bawah *ui* terus menekan mouse kiri untuk menyuruh *units* berjalan ke target tanah.
    c. *Stop*, untuk pergerakan *stop* pemain dapat menekan tombol S di *keyboard* atau menekan tombol *stop* di bawah ui untuk menyuruh *units* untuk berhenti di tempat.
    d. *Hold*, untuk pergerakan *hold* pemain dapat menekan tombol H di *keyboard* atau menekan tombol *hold* di bawah *ui* untuk menyuruh *units* untuk bertahan di tempat, dan tidak menyerang.

3. *Build*, pemain dapat memangun bangunan dengan menggunakan Trombocytes untuk membangun banyak macam bangunan. Sebagian bangunan dapat melatih pasukan cells, ada bangunan untuk mengumpulkan *resource*, ada bangunan untuk menambah kapasitas *resource*, dan bangunan untuk menahan serangan dari musuh.

Kemenangan dapat diraih jika *quest* yang diberikan harus diselesaikan, *quest* mempunyai banyak macam, dan setiap *quest* akan berbeda untuk tiap *level game* seperti:

1. *Level* 0 *Tutorial*, *game* akan menang jika pemain berhasil menyelesaikan semua *quest tutorial* sampai akhir.
2. *Level* 1 *Exteriminate*, *game* akan menang jika pemain berhasil membasmi semua musuh yang ada di peta.
3. *Level* 2 *Defend*, *game* akan menang jika pemain berhasil menahan serangan musuh sampai waktu habis.
4. *Level* 3 *Survival*, *game survival* tidak ada kondisi menang karena pemain harus bertahan selama mungkin sampai *Nerve Center Hancur* atau pemain keluar dari *game*. *Score* akan tetap di catat jika *Nerve Center* hancur atau keluar dari *game*.

Pemain akan kalah jika *Nerve Center* hancur. Pemain dapat mencegah kekalahan, dengan membangun bangunan di dekat *Nerve Center,* jadi musuh yang mendekati tidak dapat langsung menyerang *Nerve Center*. Atau melatih banyak pasukan untuk membasmi musuh yang datang.

3.1.1. Kontrol Pemain

Pemain menggunakan mouse sebagai kontrol utama untuk bermain, selain itu untuk memudahkan kontrol pemain di berikan *shortcut keyboard* yang akan di jelaskan di tabel 3.1.

Tabel 3.1
Shortcut Keyboard

Tombol	**Aksi**
A (*Attack*)	Menyuruh *Selected Unit* / *Units* untuk menyerang target.

Tombol	**Aksi**
M (*Move*)	Menyuruh *Selected Unit* / *Units* untuk bergerak ke target.
S (*Stop*)	Menyuruh *Selected Unit* / *Units* untuk berhenti.
H (*Hold*)	Menyuruh *Selected Unit* / *Units* untuk menahan di tempat.
1-5	Merubah *Selected Units* menjadi *Group* 1-5.
ALT + 1-5	Menetapkan *Selected Units* menjadi *Group* 1-5
Left CTRL	*Toggle* antara memperlihatkan atau menyembunyikan nyawa unit.
Up Arrow	Menggerakan kamera *game* ke atas.
Down Arrow	Menggerakan kamera *game* ke bawah.
Left Arrow	Menggerakan kamera *game* ke kiri.
Right Arrow	Menggerakan kamera *game* ke kanan.
ESC	*Pause* atau *Resume game* dan menampilkan atau menyembunyikan *Menu Paused*.

3.1.2. Jenis *Cells*

Di dalam *game Cells* mengandung banyak macam *cell cell*. Nama *cell*, tipe dan informasi dari masing masing *cell* dapat di liat di tabel 3.1. Tipe *cells* terdapat banyak macam seperti:

1. Tipe *cell Unit*, karakter yang dapat digerakan.
2. Tipe *cell* Bangunan, tidak dapat bergerak tapi dapat melakukan yang lain seperti melatih atau mengumpulkan resource.
3. Tipe *cell* Teman, untuk karakter yang dimiliki oleh pemain.
4. Tipe *cell* Musuh, untuk karakter yang di gerakan sama musuh.
5. Tipe *cell Builder*, untuk karater yang dapat membangun atau memperbaiki bangunan, mengumpulkan *resource*, dan mereka akan kabur jika kena serang.

Tabel 3.2
Jenis *Cells*

Nama	**Tipe *Cell***	**Informasi**
Red Blood Cell	*Unit*, Teman, *Builder*	Tipe *builder* yang mengumpulkan *resource.*
Thrombocytes	*Unit*, Teman, *Builder*	Tipe *builder* yang dapat membangun bangunan atau memperbaiki bangunan.

Nama	**Tipe *Cell***	**Informasi**
White Blood Cell	*Unit*, Teman	Pertahanan utama sistem pertahanan tubuh.
Macrophage	*Unit*, Teman	Cell yang tebal, dan mempunyai pertahanan yang tebal untuk menahan serangan musuh.
B Cell	*Unit*, Teman	Menyediakan *damage chemical* yang mempunyai 50% *damage* tambahan terhadap bakteria.
T Cell	*Unit*, Teman	Pertahanan terakhir sistem pertahanan tubuh, mempunyai nyawa yang tebal, dan serangan yang besar.
Eosinophils	*Unit*, Teman	Menyediakan *damage viral* yang mempunyai 100% *damage* tambahan terhadap parasit.
Nerve Center	Bangunan, Teman	Bangunan utama yang memberi perintah untuk semua *cell.* Jika hancur *game* akan berakhir
Intestine	Bangunan, Teman	Bangunan untuk mengambil *resource sugar.*
Artery	Bangunan, Teman	Bangunan untuk mengambil *resouce oxygen.*
Lymph Node	Bangunan, Teman	Bangunan untuk melatih *B Cell* dan *T Cell*.
Bone Marrow	Bangunan, Teman	Bangunan untuk melatih *Trombocytes, Red Blood Cell, White Blood Cell,* dan *Eosinophils.*

Nama	**Tipe *Cell***	**Informasi**
Fat	Bangunan, Teman	Menyediakan kapasitas tambahan untuk *resources.*
Cell Wall	Bangunan, Teman	Bangunan untuk menahan serangan musuh.
Helicobacter Pylori	Musuh	Bakteria yang biasanya ditemukan di lambung yang dapat mempertipis lapisan lambung.
Streptococcus Pneumoniae	Musuh	Bakteria yang di temukan di saluran pernapasan yang dapat mengakibatkan batuk, dan susah bernafas.
Influenza	Musuh	Virus yang di temukan di saluran pernapasan, dapat berkembang biak jika mengalahkan *cell.*
Anisakis	Musuh	Parasit yang datang dari makan ikan mentah, mereka mempunyai nyawa yang tebal dan dapat gampang merusak bangunan.

3.1.3. Spesifikasi Kebutuhan Perangkat Lunak

Minimum perangkat lunak yang di butuhkan untuk menjalankan *game Cells* adalah.

1. OS: Windows 10, 64-bit versions only.
2. CPU: X64 architecture with SSE2 instruction set support.
3. GPU: DX10.
4. Unity Version: 2019.1.11f1.

3.1.4. Spesifikasi Kebutuhan Perangkat Keras

Minimum perangkat keras yang di butuhkan untuk menjalankan *game Cells* adalah.

1. Processor: Intel Core i5

2. RAM: 4 GB
3. Storage: 2 GB

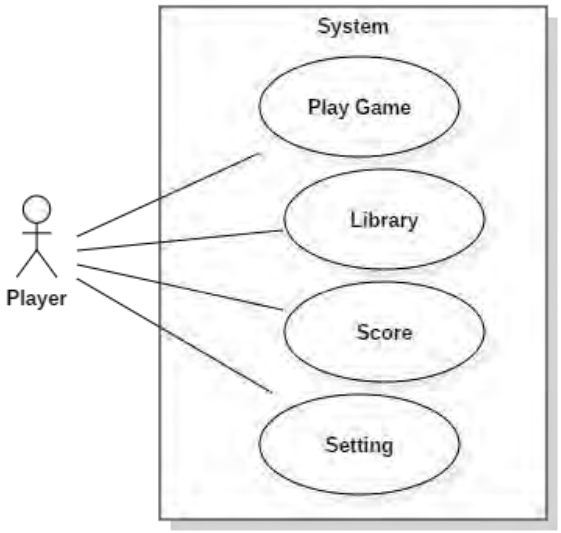
3.1.5. Analisi Kebutuhan Fungsional

Analisi kebutuhan fungsional adalah fitur fitur yang tersedia dari sistem yang akan berinteraksi antara pemain dan sistem. Bagaimana sistem berjalan dan mengelolah perintah dari pemain. Berikut adalah kebutuhan fungsi dari *game Cells*:

1. Sistem dapat nemapilkan kemenangan dan kekalahan pemain.
2. Sistem dapat menghitung total *score* dalam permainan.
3. Sistem dapat *save data*: *Level*, *Scores*, *Library*, dan *Setting*.
4. Sistem dapat memberi *quest* untuk pemain.
5. Sistem dapat *pause* semua aktifitas jika pemain membuka *menu*.
6. Sistem dapat menampilkan data *Library* yang sudah di akses.

3.2. *Use Case Diagram*

Use case diagram perangkat lunak *game real time strategy* ditunjukan pada gambar 3.1. Terdapat 4 pilihan yang tersedia di dalam *game*: *Play Game*, *Cell Library*, *View Score*, *Setting*.



Gambar 3.1
Use Case Diagram Perangkat Lunak *Game Cells*

3.3. *Scenario Diagram*

Scenario Diagram digunakan untuk menggambarkan pengguna berinteraksi oleh *game*, dan jalur jalur yang akan di tempuh oleh pemain dan sistem.

3.3.1. *Scenario Play Game*

Use Case : *Play Game*

Actor : Pemain

Description : Scenario untuk *Play Game* menggambarkan pemain saat memilih *level game* untuk di mainkan.

Precondition : *Game* sudah menampilkan Logo.

Postcondition : Menyelesaikan semua *quest*.

Tabel 3.3
Scenario Use Case Play Game: *Level* 0 *Tutorial*

Aktor	**Sistem**
1.*Player* memilih *Start*.	
	2.Menampilkan Panel *Start*.
3.*Player* memilih *Tutorial*.	
	4.Masuk ke Dalam *Scene Tutorial*.
	5.Memberikan *Quest* "*Move Camera*"
6.*Player* menggerakan *Camera* menggunakan mouse atau *arrow keyboard*.	
	7.*Quest* Berhasil Diselesaikan.
	8.Memberikan 1 *White Blood Cell*.
	9.Memberikan *Quest* "*Select Unit*".
10.*Player* memilih *unit White Blood Cell* menggunakan mouse klik kiri.	
	11.*Quest* Berhasil Diselesaikan.
	12.Memberikan *Quest* "*Move Selected Unit*".
13.*Player* menggerakan *unit* menggunakan mouse klik kanan ke tanah, atau menekan tombol "M" terus klik kiri mouse ke tanah, atau menekan tombol di *UI* terus klik kiri mouse, untuk menyuruh unit yang telah di pilih bergerak maju ke target tanah.	
	14.*Quest* Berhasil Diselesaikan.
	15.Memberikan 1 *Macrophage*.
	16.Memberikan *Quest* "*Select Multiple Unit*".
17.*Player* menahan klik kiri mouse dan di gerakan mouse, untuk membikin	

Aktor	Sistem
Selection Box, untuk *select* semua *unit* yang di dalam kotak tersebut.	
	18.*Quest* Berhasil Diselesaikan.
	19.Mengeluarkan 1 Musuh dan menyerang *Nerve Center*.
	20.*Camera* di pindahkan ke musuh.
	21.Memberikan *Quest* "*Eliminate Attacking Enemy*".
22.*Player* menggerakan *selected units* menekati musuh dengan klik kanan *mouse*.	
	23.*Selected Units* mendeteksi musuh dan bergerak mendekati musuh, setelah dekat *unit* akan mulai berkelahi.
	24.*Unit* Pemain berhasil mengalahkan musuh.
	25.*Quest* Berhasil Diselesaikan.
	26.Memberikan 1 *Trombocytes*.
	27.Memberikan Quest "*Build* 2 *Fat* & 1 *Bone Marrow*".
28.*Player* memilih *Trombocytes*.	
	29.Menampilkan *List* Bangunan yang dapat di bangun.
30.*Player* menekan tombol *ui* untuk membangun 2 *Fat*, dan 1 *Bone Marrow*.	
	31.Bangunan selesai di bangun semua.
	32.*Quest* Berhasil Diselesaikan.
	33.Memberikan *Quest* "*Train* 2 *White Blood Cell*".
34.*Player* memilih *Bone Marrow*	
	35.Menampilkan *List Unit* yang dapat di latih.
36.*Player* menekan tombol *ui White Blood Cell* 2 kali.	
	37.Mulai melatih *White Blood Cell*.
	38.Latihan selesai, mengeluarkan 1 *White Blood Cell*.
	39.Mulai melatih *White Blood Cell* ke dua.
	40.Latihan selesai, mengeluarkan 1 *White Blood Cell*.
	41.*Quest* Berhasil Diselesaikan.
	42.Memberikan *Quest* "*Defend Nerve Center*".
43.*Player* menahan serangan musuh.	
	44.Berhasil menahan serangan.
	45.*Quest* Berhasil Diselesaikan.
	46.Jika *level* pemain 0 *Level* Pemain di tambah menjadi 1.
	47.Keluar dari *Scene* dan balik ke *Main Menu*.

Tabel 3.4
Scenario alternatif *use case Play Game* 1: *Level* 0 *Tutorial Unit* Kalah Semua

Aktor	**Sistem**
1-23 Sama dengan scenario normal *Play Game*.	
	24.*Unit* Pemain Terkalahkan semua atau *Nerve Center* hancur.
	25.*Quest* Gagal Diselesaikan.
	26.Menampilkan *Game Over* Panel.
	27.Keluar Dari *Scene.*

Tabel 3.5
Scenario alternatif *use case Play Game* 2: *Level* 0 *Tutorial Nerve Center* Hancur

Aktor	**Sistem**
1-43 Sama dengan scenario normal *Play Game*.	
	44.*Nerve Center* hancur.
	45.*Quest* Gagal Diselesaikan.
	46.Menampilkan *Game Over* Panel.
	47.Keluar Dari *Scene.*

Tabel 3.6
Scenario alternatif *use case Play Game* 3: *Level* 0 *Tutorial* Keluar Dari *Game*

Aktor	**Sistem**
Sekitar 1-43 scenario normal *Play Game*.	
44.*Player* menekan tombol *Menu* atau menekan *ESC*.	
	45.Menampilkan *Menu Paused.*
	46.*Load* data *setting volume*.
47.*Player* menekan tombol *Exit*.	
	48.Keluar dari *Scene*, dan masuk dalam *scene Main Menu*.

Tabel 3.7
Scenario alternatif *use case Play Game* 4: *Level* 0 *Tutorial* Mengubah *Setting Volume*

Aktor	**Sistem**
1-45 Sama dengan scenario normal *Play Game*.	
46.*Player* merubah *setting Volume Music*.	
	47.Merubah *volume music*.
48.*Player* merubah *setting Volume Sound*.	
	49.Merubah *volume sound*.
50.*Player* menekan tombol *Apply*.	
	51.*Save volume music* dan *sound*.
52.*Player* menekan tombol *Resume*.	
	53.Lanjut bermain.

Tabel 3.8
Scenario alternatif *use case Play Game* 5: Bermain di *Level* 1 *Exterminate*

Aktor	**Sistem**
1-2 Sama dengan scenario normal *Play Game*.	
3.*Player* memilih *Exterminate*.	
	4.Masuk ke Dalam *Scene Exterminate*.
	5.Memberikan *Quest* "*Exterminate*".
	6.Memberikan *Enemy Counter*.
	7.*Spawn* 1 *Nerve Center*, 2 *White Blood Cell*, 2 *Red Blood Cell*, 2 *Trombocytes*.
8.Menggerakan *Red Blood Cell* untuk mengumpulkan resource.	
	9.Menambah *Resource*.
10.Menggerakan *Trombocytes* untuk membangun *Bone Marrow*.	
	11.*Bone Marrow* berhasil di bangun.
12.*Player* Memilih *Bone Marrow*	
	13.Menampilkan *List Train Units Bone Marrow*.
14.*Player* Memilih *White Blood Cell Untuk* di Latih.	
	15.Melatih *White Blood Cell*.
	16.*White Blood Cell* Selesai di Latih.
17.*Player* Menggerakan Semua *White Blood Cell* Untuk Mengalahkan Musuh.	
	18.Mengalahkan Musuh Mengurangi *Enemy Counter*, dan Tambah *Score*.
19.Player Melanjutkan Menggumpulkan *Resource*, Melatih Pasukan, dan Lanjut Mengalahkan Musuh.	
	20.*Enemy Counter* Sampai Kosong
	21.*Quest* Berhasil Diselesaikan.
	22.*Score* di Tambah dari *Timer*, Semakin Cepat Semakin Besar.
	23.*Score* Dicatat dan di *Save*.
	24.*Level* di Tambah dan di *Save*.
	25.Keluar Dari *Scene.*

Tabel 3.9
Scenario alternatif *use case Play Game* 6: *Level* 1 *Exterminate Nerve Center* Hancur

Aktor	**Sistem**
1-19 Sama dengan scenario 5 *Play Game*.	
	20.*Nerve Center* Hancur.
	21.*Quest* Gagal Diselesaikan.
	22.Keluar Dari *Scene.*

Tabel 3.10
Scenario alternatif *use case Play Game* 7: *Level* 1 *Exterminate* Keluar Dari *Game*

Aktor	**Sistem**
Sekitar 1-19 scenario 5 *Play Game*.	

Aktor	Sistem
20.*Player* menekan tombol *Menu* atau menekan *ESC*.	
	21.Menampilkan *Menu Paused.*
	22.*Load* data *setting volume*.
23.*Player* menekan tombol *Exit*.	
	24.Keluar dari *Scene*, dan masuk dalam *scene Main Menu*.

Tabel 3.11
Scenario alternatif *use case Play Game* 8: *Level* 1 *Exterminate* Mengubah *Setting* Volume

Aktor	Sistem
1-19 Sama dengan scenario 5 *Play Game*.	
20.*Player* merubah *setting Volume Music*.	
	21.Merubah *volume music*.
22.*Player* merubah *setting Volume Sound*.	
	23.Merubah *volume sound*.
24.*Player* menekan tombol *Apply*.	
	25.Save *volume music* dan *sound*.
26.*Player* menekan tombol *Resume*.	
	27.Lanjut bermain.

Tabel 3.12
Scenario alternatif *use case Play Game* 9: *Level* 2 *Defend*

Aktor	Sistem
1-2 Sama dengan scenario normal *Play Game*.	
3.*Player* memilih *Defend*.	
	4.Masuk ke Dalam *Scene Defend*.
	5.Memberikan *Quest* "*Defend*".
	6.Memberikan *Defend Timer*.
	7.Spawn 1 *Nerve Center*, 2 *White Blood Cell*, 2 *Red Blood Cell*, 2 *Trombocytes*.
8.Menggerakan *Red Blood Cell* untuk mengumpulkan *resource*.	
	9.Menambah *Resource*.
10.Menggerakan *Trombocytes* untuk membangun *Bone Marrow*.	
	11.*Bone Marrow* berhasil di bangun.
12.*Player* Memilih *Bone Marrow*	
	13.Menampilkan *List Train Units Bone Marrow*.
14.*Player* Memilih *White Blood Cell* Untuk di Latih.	
	15.Melatih *White Blood Cell*.
	16.*White Blood Cell* Selesai di Latih.
17.*Player* Menggerakan Semua *White Blood Cell Untuk* Menahan Musuh.	

Aktor	Sistem
	18.Muncul Musuh Dari Semua Arah, Bergerak Mendekati *Nerve Center* dan Menyerang Semua yang di Dekat Mereka.
	19.Mengalahkan Musuh Menambah *Score*.
20.*Player* Melanjutkan Menggumpulkan *Resource*, Melatih Pasukan Untuk Menahan Serangan.	
	21.*Timer* Habis.
	22.*Quest* Berhasil Diselesaikan.
	23.*Score* Dicatat dan di *Save*.
	24.*Level* di Tambah dan di *Save*.
	25.Keluar Dari *Scene.*

Tabel 3.13
Scenario alternatif *use case Play Game* 10: *Level* 2 *Defend Nerve Center* Hancur

Aktor	Sistem
1-20 Sama dengan scenario 9 *Play Game*.	
	21.*Nerve Center* Hancur.
	22.*Quest* Gagal Diselesaikan.
	23.Keluar Dari *Scene.*

Tabel 3.14
Scenario alternatif *use case Play Game* 11: Level 2 Defend Keluar Dari *Game*

Aktor	Sistem
Sekitar 1-20 scenario 9 *Play Game*.	
21.*Player* menekan tombol *Menu* atau menekan *ESC*.	
	22.Menampilkan *Menu Paused.*
	23.*Load* data setting *volume*.
24.*Player* menekan tombol *Exit*.	
	25.Keluar dari *Scene*, dan masuk dalam *scene Main Menu*.

Tabel 3.15
Scenario alternatif *use case Play Game* 12: *Level* 2 *Defend* Mengubah *Setting Volume*

Aktor	Sistem
1-20 Sama dengan scenario 5 *Play Game*.	
21.*Player* merubah *setting Volume Music*.	
	22.Merubah *volume music*.
23.*Player* merubah *setting Volume Sound*.	
	24.Merubah *volume sound*.
25.*Player* menekan tombol *Apply*.	
	26.*Save volume music* dan *sound*.
27.*Player* menekan tombol *Resume*.	
	28.Lanjut bermain.

Tabel 3.16
Scenario alternatif *use case Play Game* 13: *Level* 3 *Survival*

Aktor	**Sistem**
1-2 Sama dengan scenario normal *Play Game*.	
3.Player memilih *Survival*.	
	4.Masuk ke Dalam *Scene Survival*.
	5.Memberikan *Quest* "*Survive*".
	6.Memberikan *Survive Timer*.
	7.Spawn 1 *Nerve Center*, 2 *White Blood Cell*, 2 *Red Blood Cell*, 2 *Trombocytes*.
8.Menggerakan *Red Blood Cell* untuk mengumpulkan *resource*.	
	9.Menambah *Resource*.
10.Menggerakan *Trombocytes* untuk membangun *Bone Marrow*.	
	11.*Bone Marrow* berhasil di bangun.
12.*Player* Memilih *Bone Marrow*	
	13.Menampilkan *List Train Units Bone Marrow.*
14.*Player* Memilih *White Blood Cell* Untuk di Latih.	
	15.Melatih *White Blood Cell*.
	16.*White Blood Cell* Selesai di Latih.
17.*Player* Menggerakan Semua *White Blood Cell* Untuk Menahan Musuh.	
	18.Muncul Musuh Dari Semua Arah, Bergerak Mendekati *Nerve Center* dan Menyerang Semua yang di Dekat Mereka.
	19.Mengalahkan Musuh Menambah *Score*.
20.*Player* Melanjutkan Menggumpulkan *Resource*, Melatih Pasukan Untuk Menahan Serangan.	
	21.Setiap 5 Menit Musuh Bertambah Kuat.
	22.*Nerve Center* Hancur.
	23.*Score* Dicatat dan di *Save*.
	24.Keluar Dari *Scene.*

Tabel 3.17
Scenario alternatif *use case Play Game* 15: *Level* 3 *Survival* Keluar Dari *Game*

Aktor	**Sistem**
Sekitar 1-21 scenario 13 *Play Game*.	
22.*Player* menekan tombol *Menu* atau menekan *ESC*.	
	23.Menampilkan *Menu Paused.*
	24.*Load* data *setting volume*.
25.*Player* menekan tombol *Exit*.	
	26.Keluar dari *Scene*, dan masuk dalam *scene Main Menu*.

Tabel 3.18
Scenario alternatif *use case Play Game* 16: *Level* 3 *Survival* Mengubah *Setting Volume*

Aktor	**Sistem**
1-20 Sama dengan scenario 13 *Play Game*.	
21.*Player* merubah *setting Volume Music*.	
	23.Merubah *volume music*.
24.*Player* merubah *setting Volume Sound*.	
	25.Merubah *volume sound*.
26.*Player* menekan tombol *Apply*.	
	27.*Save volume music* dan *sound*.
28.*Player* menekan tombol *Resume*.	
	29.Lanjut bermain.

3.3.2. *Scenario Library*

Use Case : *Library*

Actor : Pemain

Description : *Scenario* unuk melihat informasi tentang semua *cell* yang ada di dalam *game*, nama dan status masing masing *cell*.

Precondition : *Game* sudah menampilkan Logo.

Postcondition : Kembali ke *Main Menu*.

Tabel 3.19
Scenario Use Case Library

Aktor	**Sistem**
1.Player memilih *Library*.	
	2.Menampilkan tab *Library.*
	3.*Load* data *Library*.
	4.Menampilkan Data Pernah di Akses.
6.Memilih *Exit*.	

Tabel 3.20
Scenario alternatif *use case Library* 1: Data Tidak Pernah di Akses

Aktor	**Sistem**
1-3 Sama dengan scenario normal *Library*.	
	4.Menyembunyikan Data Tidak Pernah di Akses.
3.Memilih *Exit*.	

3.3.3. *Scenario Scores*

Use Case : *Scores*

Actor : Pemain

Description : *Scenario* unuk menggambarkan pemain melihat *score* yang mereka telah dapatkan dari game yang mereka telah mainkan.

Precondition : *Game* sudah menampilkan Logo.

Postcondition : Kembali ke *Main Menu*.

Tabel 3.21
Scenario Use Case View Scores

Aktor	**Sistem**
1.Player memilih *Scores*.	
	2.Menampilkan tab *Score*.
	3.*Load* data *scores*.
	4.Menampilkan data *scores*.
5.Player memilih *Exit*.	

3.3.4. *Scenario Setting*

Use Case : *Setting*

Actor : Pemain

Description : *Scenario* untuk menggambarkan pemain mengubah suara *volume game* sesuai kemauan pemain.

Precondition : *Game* sudah menampilkan Logo.

Postcondition : Kembali ke *Main Menu*.

Tabel 3.22
Scenario Use Case Setting

Aktor	**Sistem**
1.*Player* memilih Setting.	
	2.Menampilkan tab *Setting*.
3.*Player* merubah *setting Volume Music*.	
	4.Merubah *volume music*.
5.*Player* merubah *setting Volume Sound*.	
	6.Merubah *volume sound*.
7.*Player* menekan tombol *Apply*.	
	8.Save *volume music* dan *sound*.
9.*Player* menekan tombol *Exit*.	

Tabel 3.23
Scenario alternatif *use case Setting* 1: Keluar Dari *Setting* Tanpa *Apply*

Aktor	**Sistem**
1.*Player* memilih *Setting*.	
	2.Menampilkan tab *Setting*.
3.*Player* merubah *setting Volume Music*.	
	4.Merubah *volume music*.

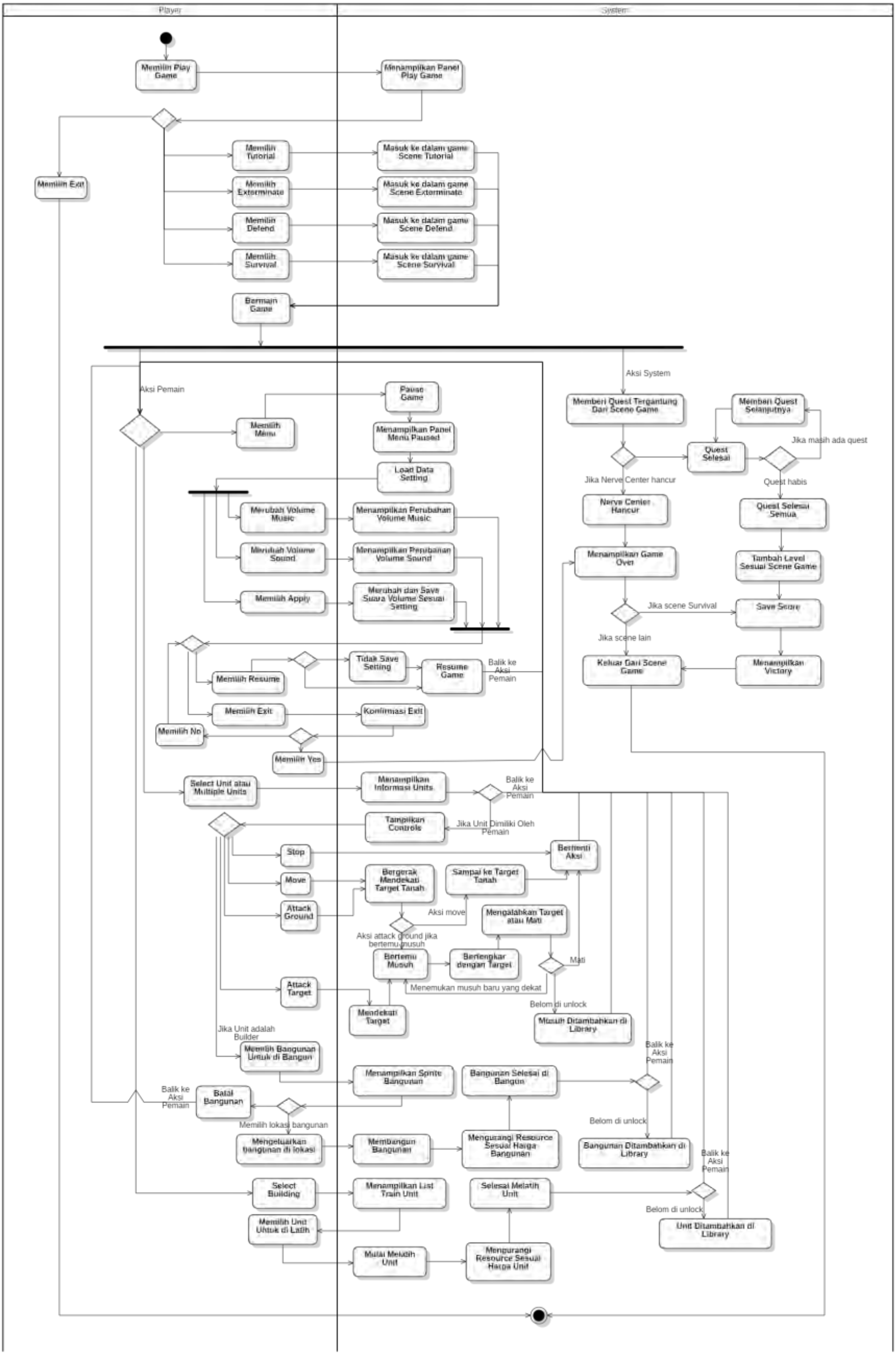
5.*Player* merubah *setting Volume Sound*.	
	6.Merubah *volume sound*.
7.*Player* menekan tombol *Exit*.	
	8.Gagal *save volume music* dan *sound*.

3.4. *Activity Diagram*

Activity diagram digunakan untuk menggambarkan urutan kegiatan perangkat lunak dan interaksi antar pemain dan sistem dari mula sampai akhir.

3.4.1. *Activity Diagram Play Game*

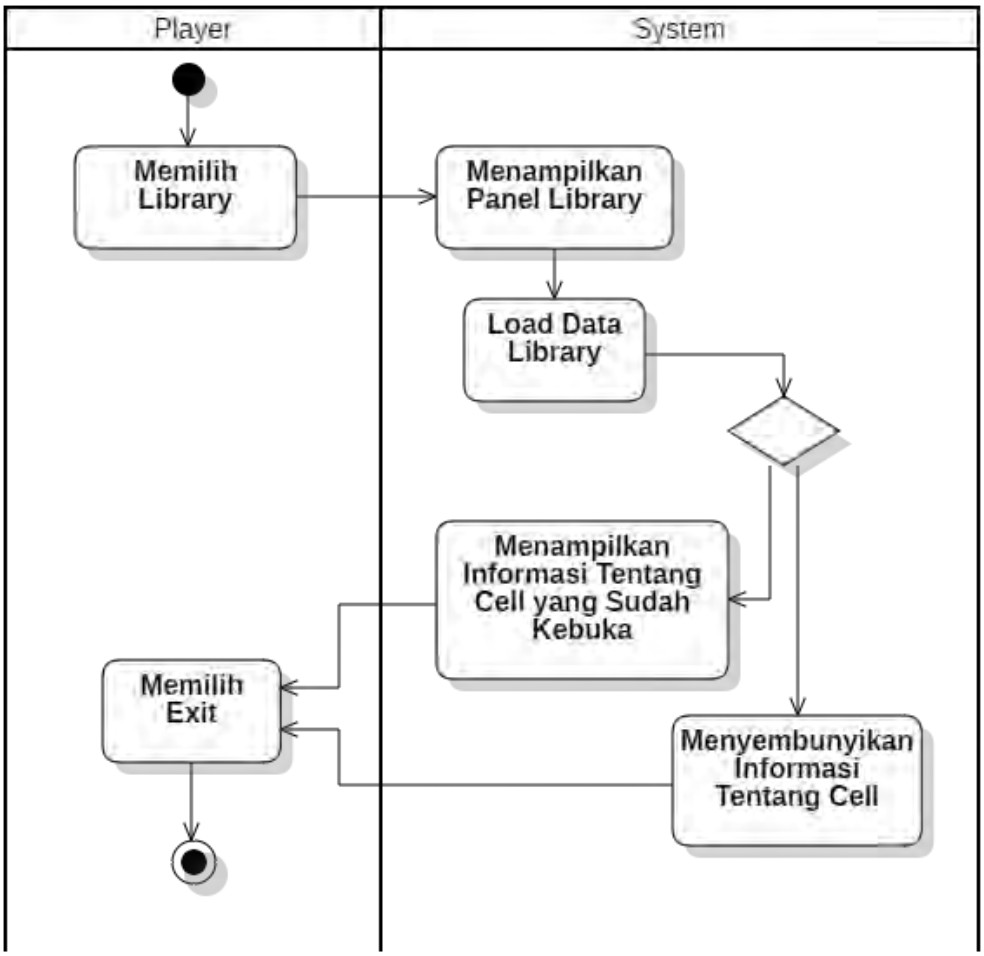
Activity diagram Play Game di tunjukan pada gambar 3.2. Saat pemain menggunakan fitur ini, sistem akan menampilkan 4 mode permainan: *Tutorial*, *Exterminate*, *Defend*, dan *Survival*. Di dalam *Tutorial* pemain dapat belajar bagaimana cara bermain *game Cells*, bagaimana cara menggerakan kamera, menggerakan pasukan atau prajurit, dan cara membangun bangunan untuk melatih prajurit baru, atau membangun tembok untuk menahan serangan musuh. Di dalam *exterminate* pemain di suruh untuk membasmi semua musuh yang ada di dalam map. Di dalam *Defend* pemain di suruh untuk menahan serangan musuh yang keluar dari semua arah selama *timer* berjalan. Di dalam *Survival* pemain harus bertahan selama mungkin, sedangkan musuhnya bertambah kuat semakin lama permainan bermain. Pemain juga dapat keluar dari *game* jika merasa sudah lelah, tetapi *score* tidak akan di *save*, kecuali *survival* dimana pemain jika keluar *score* akan tetap di *save*. Jika pemain mengalahkan musuh, pemain dapat melihat informasi tentang musuhnya di *library*, mulai dari status mereka, informasi, nyawa, *armour* mereka, dan lain-lainnya. Jika *unit* selesai di latih, pemain dapat meliha juga informasinya di *library.*



Gambar 3.2
Activity Diagram "*Start Game*"

3.4.2. *Activity Diagram Library*

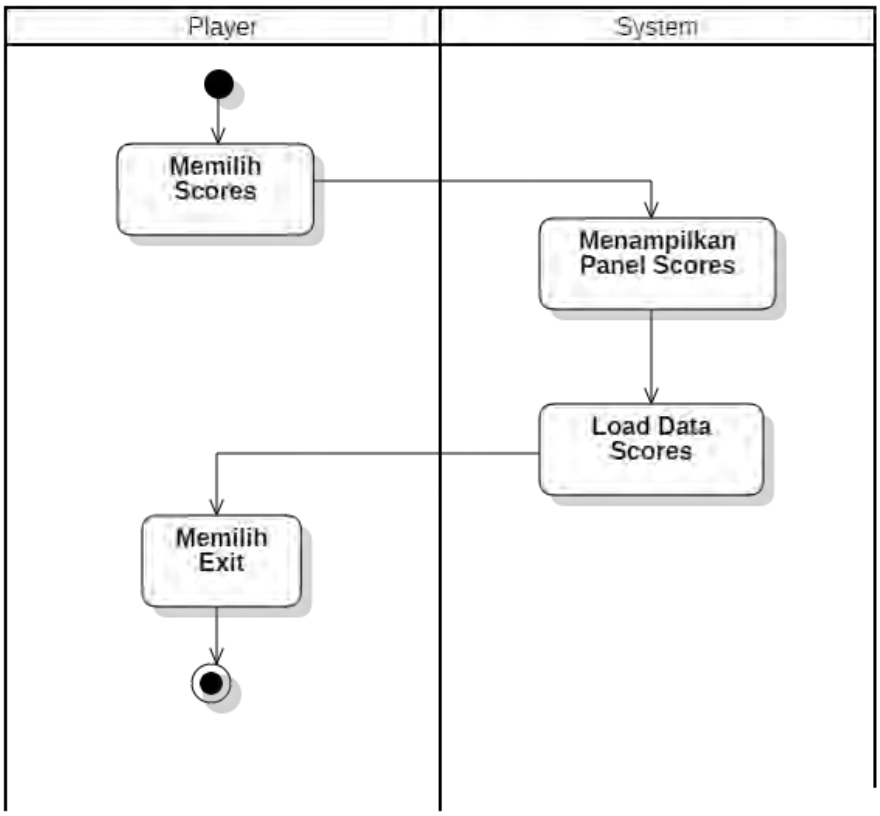
Activity diagram Library di tunjukan pada gambar 3.3. Saat pemain menggunakan fitur ini pemain dapat mengetahui tentang *cell cell* yang ada di dalam *game*. Dan informasi seperti nyawa, *armour*, *damage*, *movement speed* dan lain lain.



Gambar 3.3
Activity Diagram "*Library*"

3.4.3. *Activity Diagram Scores*

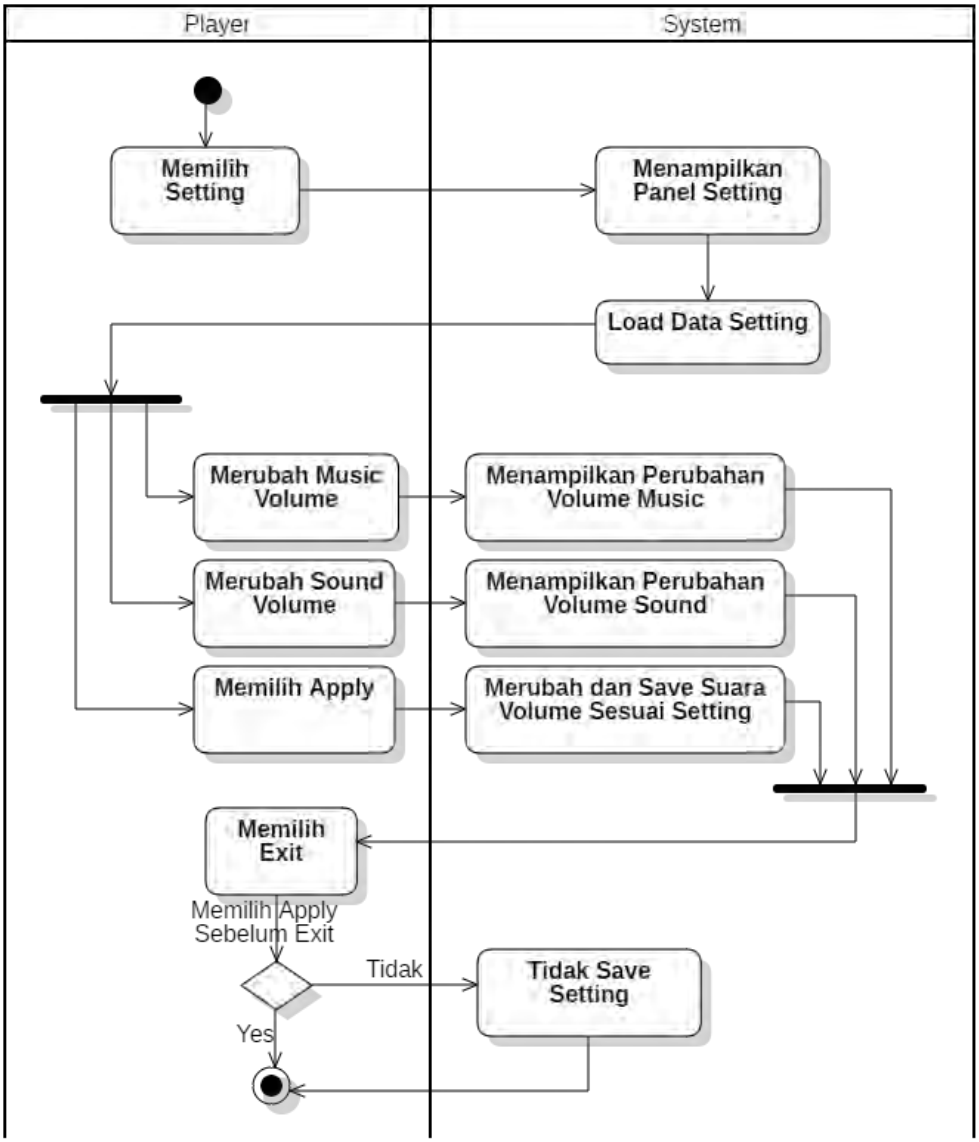
Activity diagram Scores di tunjukan pada gambar 3.4. Saat pemain menggunakan fitur ini, sistem akan menampilkan tab *Score Board* dimana pemain dapat melihat *score* yang mereka telah raih dari *game* yang pemain mainkan.



Gambar 3.4
Activity Diagram "*Scores*"

3.4.4. *Activity Diagram Setting*

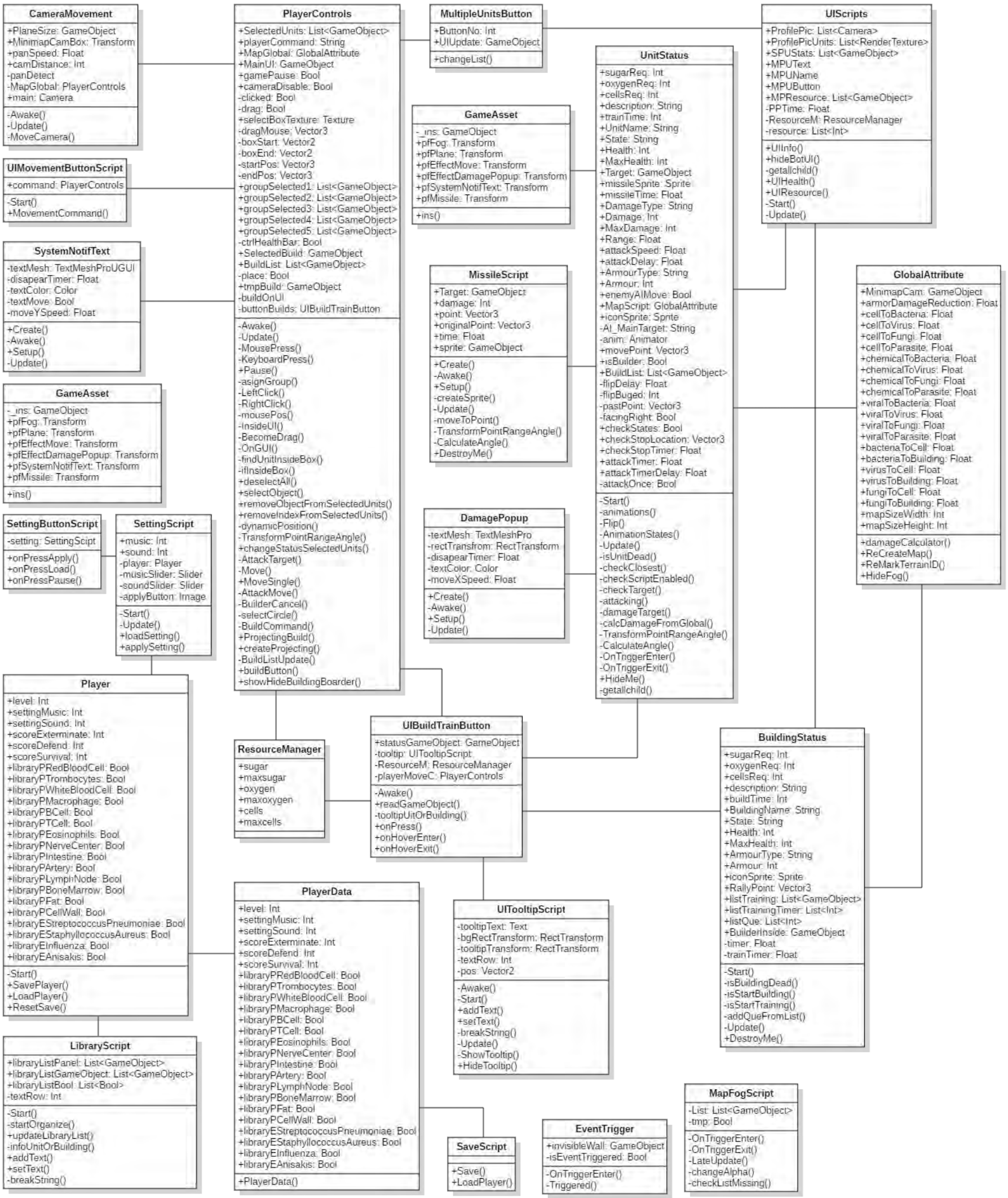
Activity diagram Setting di tunjukan pada gambar 3.5. Saat pemain menggunakan fitur ini, sistem akan menampilkan tab *Setting* dimana pemain dapat merubah suara *volume game*, mau dari *BGM* (*Back Ground Music*) atau suara yang dikeluarakan.



Gambar 3.5
Activity Diagram "*Setting*"

3.5. *Class Diagram*

Class diagram digunakan untuk menggambarkan kelas-kelas, dan realasi antar kelas yang terdapat di dalam *game Cells*.

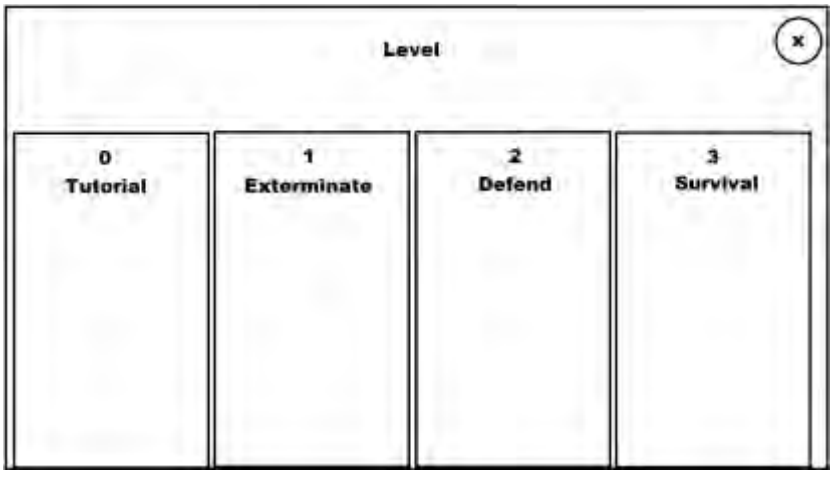


Gambar 3.6
Class Diagram Perangkat Lunak *Game Cells*

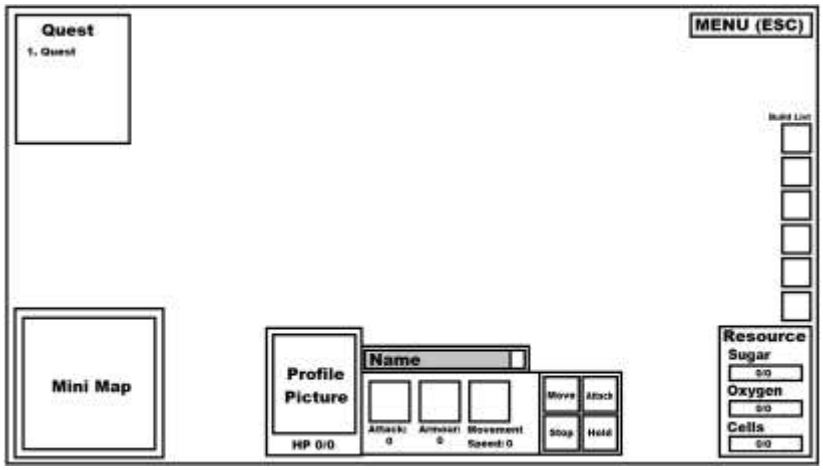
3.6. Rancangan Antarmuka

Rancangan antarmuka pada perangkat lunak *game real time strategy* digambarkan sebagai gambar 3.7 merupakan tampilan *main menu* ketika *game* dijalankan. Pemain dapat memilih 4 pilihan untuk memulai *game*, melihat perpustaka, melihat *score*, atau merubah suara *volume game* dengan *setting.* Dan gambaran untuk setiap pilihan dapat di lihat sebagai gambar 3.8 untuk *Play Game*, gambar 3.9 untuk *Library*, gambar 3.10 untuk *Scores*, gambar 3.11 untuk *Setting.*

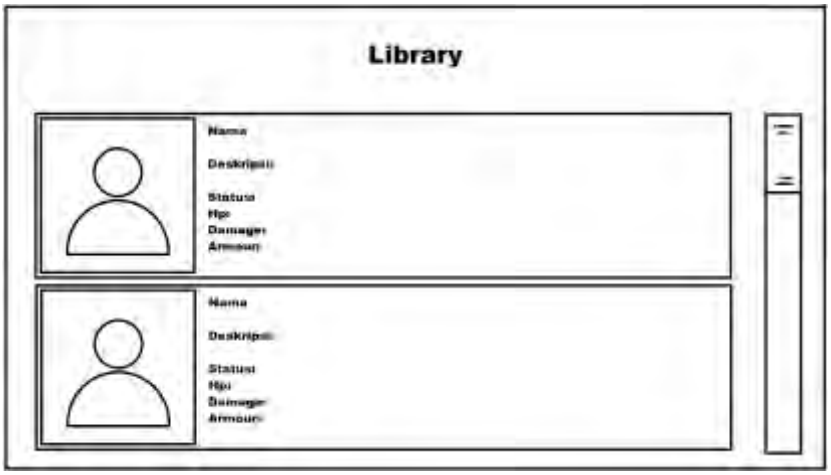
Gambar 3.7
Tampilan *Main Menu*



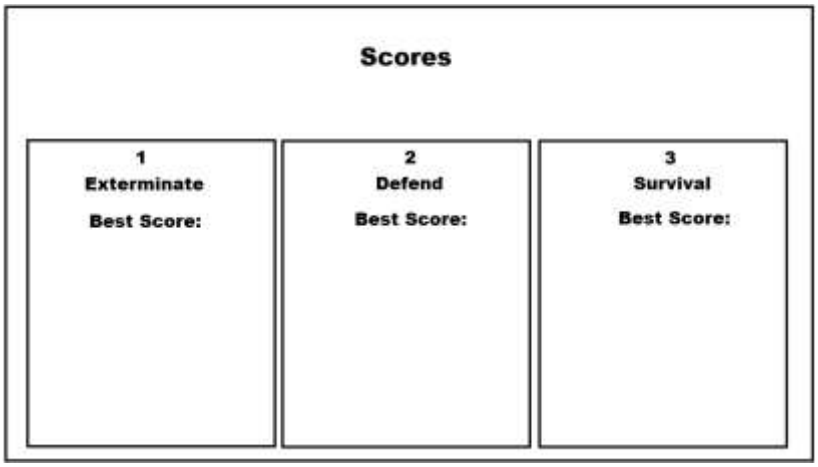
Gambar 3.8
Tampilan *Play Game*



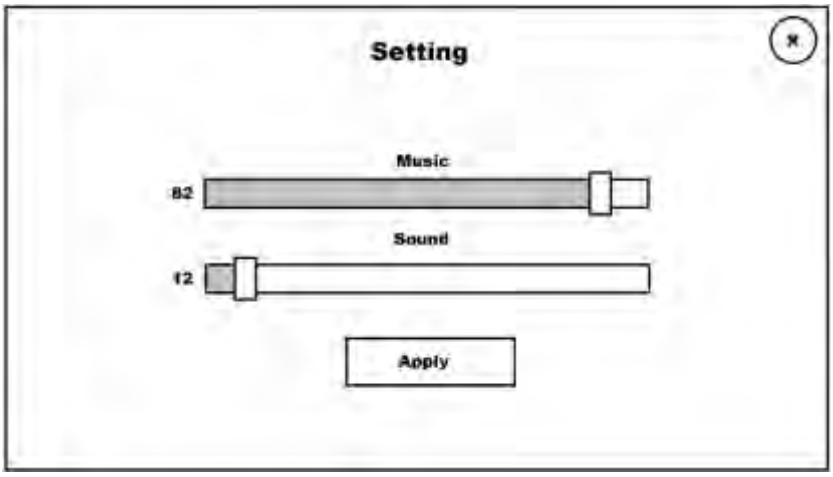
Gambar 3.9
Tampilan Dalam *Game*



Gambar 3.10
Tampilan *Cell Library*



Gambar 3.11
Tampilan *Score*



Gambar 3.12
Tampilan *Setting*

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1. Spesifikasi Kebutuhan Perangkat Keras

Spesifikasi Kebutuhan Perangkat Keras merupakan rinican tentang perangkat keras yang di butukan untuk menjalankan aplikasi dengan optimal. Perangkat keras yang di butuhkan untuk menjalankan aplikasi dijelaskan di table 4.1.

Tabel 4.1
Spesifikasi Perangkat Keras

Perangkat Keras	**Spesifikasi Perangkat Keras**
Processor	Intel Core i7
VGA	NVIDIA GeForce GTX 1050 Ti
RAM	8 GB
OS	Windows 10, 64-bit

4.2. Penjelasan *Menu*

Ketika pemain memulai *game* mereka akan di tampilkan pilihan untuk bermain sebagai berikut adalah penjelasan tentang *main menu*:

1. *Play Game* dimana pemain dapat memilih memilih untuk mulai bermain *game*, pemain dapat memilih *game level* untuk dimainkan:
    a. *Tutorial* pemain dapat memilih *level tutorial* untuk mengetahui bagai mana cara bermain *game Cells*, mulai dari cara menggerakan unit, menyerang, membangun bangunan, dan lain lain.
    b. *Exterminate* dimana misi pemain untuk membasmi semua bakteria, virus, dan parasit untuk menang.
    c. *Defend* dimana pemain harus bertahan selama waktu berjalan, dan akan menang jika pemain bertahan sampau waktu habis.
    d. *Survival* dimana pemain akan bertahan selama mungkin, selagi musuh bertambah kuat selama *game* berjalan.
2. *Library* dimana semua informasi semua cell di dalam *game*, mau dari nyawa, armor, kecepatan serang, jarak serangan, dan lain lain. Pemain harus bermain untuk menampilkan informasi tentang *cells* pemain atau musuh.

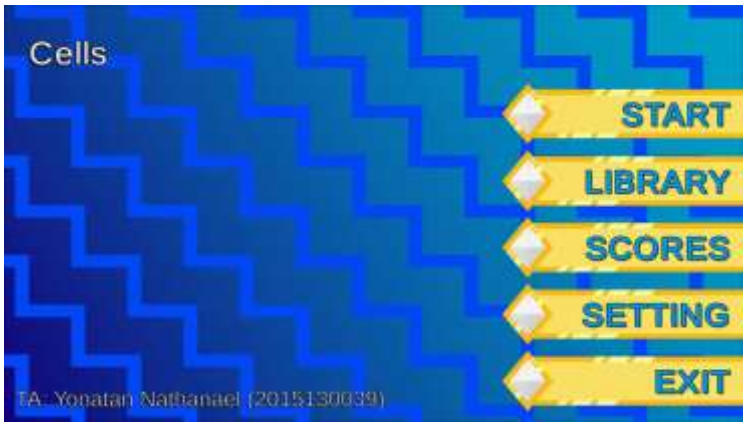
3. *Scores* dimana pemain dapat melihat *score* yang mereka telah raih dari *game* masing masing.
4. *Setting* dimana pemain dapat merubah *volume* suara *music*, dan *sound*.

4.3. Pengujian Antar Muka

4.3.1. Antar Muka *Main Menu*

Tampilan antar muka *Main Menu*, menampilkan antar muka dari *game Cells*. Tampilan antar muka *Main Menu Cells* akan muncul pertama kali saat pemain masuk ke dalam *game*. Di dalam antar muka *main menu* terdapat 5 tombol, yaitu memulai *game*, *library*, *score*, *setting*, dan *exit*.

1. *Play Game*, terletak di kanan paling atas untuk menampilkan panel *Play Game*.
2. Library, terletak di kanan di baris ke dua untuk menampilkan informasi tentang semua *cells* yang ada di dalam *game*.
3. *Score*, terletak di kanan di baris ke tiga untuk menampilkan hasil *score game* untuk masing masing *level*.
4. *Setting*, terletak di kanan di baris ke empat untuk mengubah *setting* suara.
5. *Exit*, terletak di kanan di baris terakhir untuk keluar dari *game*.



Gambar 4.1
Tampilan *Main Menu*

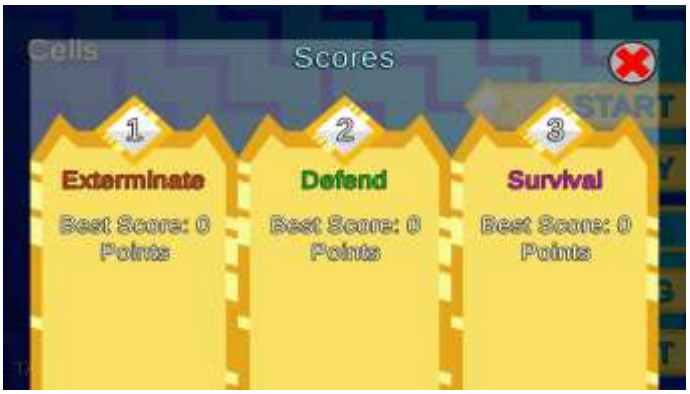
4.3.2. Antar Muka *Library*

Tampilan antar muka *Library* akan di tampilkan jika pemain menekan tombol *Library* terletak di kanan baris ke dua. Di dalam antar muka ini pemain dapat melihat informasi tentang semua cells yang ada di dalam *game*, seperti *HP*, *Damage*, *Movement Speed*, *Damage Type*, dan lain lain. Pemain dapat menutup Library *Menu* dengan menekan tombol *close*.

Gambar 4.2
Tampilan *Library Menu*

4.3.3. Antar Muka *Scores*

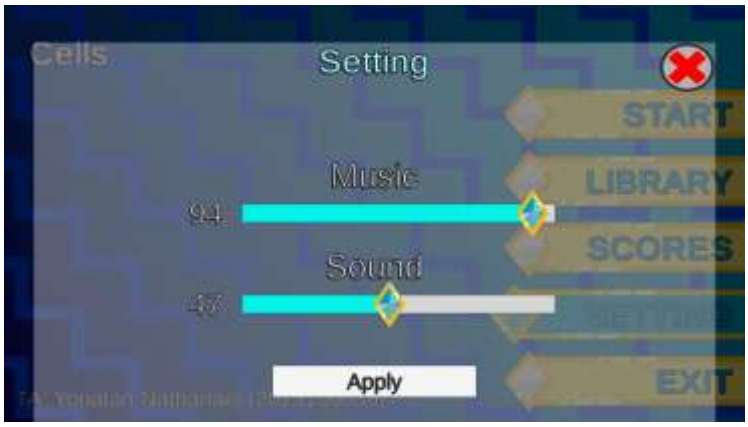
Tampilan antar muka *Scores* akan di tampilkan jika pemain menekan tombol *Scores* terletak di kanan baris ke tiga. Di dalam antar muka ini pemain dapat melihat hasil *score* dari *game* pemain.



Gambar 4.3
Tampilan *Scores Menu*

4.3.4. Antar Muka *Setting*

Tampilan antar muka *Setting* akan di tampilkan jika pemain menekan tombol *Setting* terletak di kanan baris ke empat. Di dalam antar muka ini pemain dapat merubah *setting game*.



Gambar 4.4
Tampilan *Setting Menu*

4.3.5. Antar Muka *Game*

Tampilan antar muka *game*, merupakan scene di dalam *game tutorial*, merupakan scene di mana pemain telah menekan tombol *play game* setelah itu memilih *level* 0 *tutorial*. Dalam *game* pemain dapat menggerakan *cells* seperti *White Blood Cell*, *Marophage*, *Thrombocytes*, dan lain lain. Untuk melawan dan membasmi musuh seperti bakteri, virus, dan parasit yang berada di dalam tubuh manusia.

Gambar 4.5
Tampilan *Game*

4.3.6. Antar Muka *Menu Paused*

Tampilan antar muka *menu paused* di tampilkan jika pemain menekan *keyboard escape* atau menekan tombol *menu* (*esc*) di dalam *game*, terletak di kanan atas. Di dalam *menu paused*, *game* tidak akan berjalan, dan pemain dapat mengubah *setting* suara, *resume* balik ke *game*, atau keluar dari *game* dan balik ke *main menu*.

Gambar 4.6
Tampilan *Menu Paused*

4.3.7. Antar Muka *Game Over*

Antar muka *game over* akan di tampilkan jika Nerve Center hancur, dan akan di berikan score total dari level tersebut.

Gambar 4.7
Tampilan *Game Over*

4.4. Pengujian Fungsi

Pengujian fungsi pada *game Cells* menggunakan metode *black box*. Pengujian menggunakan *black box* untuk memeriksa fungsi dari *game*.

1. *Main Menu*

Tabel 4.2
Pengujian scene *Main Menu*

No.	Pengujian	Tujuan	Hasil Diharapkan	Hasil Uji
1.	Menekan tombol *Start*.	Membuka panel *Start*.	Membuka panel *Start*.	OK
2.	Menekan tombol *Library*.	Membuka panel *Library*.	Memuka panel *Library*.	OK
3.	Menekan tombol *Scores*.	Membuka panel *Scores*.	Membuka panel *Scores*.	OK
4.	Menekan tombol *Setting*.	Membuka panel *Setting*.	Membuka panel *Setting*.	OK
5.	Menekan tombol *Exit.*	Keluar dari *Game Cells*.	Keluar dari *Game Cells*.	OK

Berdasarkan hasil dari pengujian fungi *Main Menu* pada tabel 4.2 dapat di ambil kesimpulan bahwa semua hasil yang di harapkan sesuai tujuan.

2. Panel *Play Game*

Tabel 4.3
Pengujian *Play Game*

No.	Pengujian	Tujuan	Hasil Diharapkan	Hasil Uji
1.	Tombol *Defend* dimatikan Jika *Level* < 2.	Tidak dapat di pencet.	Tidak dapat di pencet.	OK
2.	Tombol *Survival* dimatikan Jika *Level* < 3.	Tidak dapat di pencet.	Tidak dapat di pencet.	OK
3.	Menekan tombol *Tutorial*.	Membuka *Scene Tutorial*.	Membuka *Scene Tutorial*.	OK
4.	Menekan tombol *Exterminate*.	Membuka *Scene Exterminate*.	Membuka *Scene Exterminate*.	OK
5.	Menekan tombol *Defend*.	Membuka *Scene Defend*.	Membuka *Scene Defend*.	OK
6.	Menekan tombol *Survival*.	Membuka *Scene Survival*.	Membuka *Scene Survival*.	OK
7.	Menekan tombol *Exit*.	Menutup panel *Start*.	Menutup panel *Start*.	OK
8.	Menekan tombol *Menu* (*ESC*).	Membuka panel *Menu Paused*.	Membuka panel *Menu Paused*.	OK
9.	Menekan tombol *Apply* di *Menu Paused*.	*Save* Data *Setting Volume Music* & *Sound*.	*Save* Data *Setting Volume Music* & *Sound*.	OK

No.	Pengujian	Tujuan	Hasil Diharapkan	Hasil Uji
10.	Menekan tombol *Resume* (*ESC*) di *Menu Paused*.	Menutup panel *Menu Paused*.	Menutup panel *Menu Paused*.	OK
11.	Menekan tombol *Exit* di *Menu Paused*.	Membuka panel *Confrim Exit*.	Membuka panel *Confrim Exit*.	OK
12.	Menekan tombol *Yes* di panel *Confirm Exit*.	Keluar dari *Scene Game* balik ke *Scene Main Menu*.	Keluar dari *Scene Game* balik ke *Scene Main Menu*.	OK
13.	Menekan tombol *No* di panel *Confirm Exit*.	Menutup panel *Confirm Exit* balik ke panel *Menu Paused*.	Menutup panel *Confirm Exit* balik ke panel *Menu Paused*.	OK
14.	Menekan tombol *Yes* di Panel *Confirm Exit*.	*Save Score* jika *Score* lebih besar dari *Best Score Level* tersebut.	*Save Score* jika *Score* lebih besar dari *Best Score Level* tersebut.	OK
15.	Menekan tombol *Exit* di panel *Game Over*.	Keluar dari Scene *Game* balik ke Scen Main Menu, dan *Save Score* jika *Score* lebih besar dari *Best Score Level* tersebut.	Keluar dari Scene *Game* balik ke Scen Main Menu, dan *Save Score* jika *Score* lebih besar dari *Best Score Level* tersebut.	OK
16.	Menekan tombol *Attack*.	Menyuruh semua *selected units* untuk menyerang target.	Menyuruh semua *selected units* untuk menyerang target.	OK
17.	Menekan tombol *Move*.	Menyuruh semua *selected units* untuk bergerak menuju target.	Menyuruh semua *selected units* untuk bergerak menuju target.	OK
18.	Menekan tombol *Stop*.	Menyuruh semua *selected units* untuk berhenti.	Menyuruh semua *selected units* untuk berhenti.	OK
19.	Menekan tombol *Hold*.	Menyuruh semua *selected units* untuk menahan diri di tempat.	Menyuruh semua *selected units* untuk menahan diri di tempat.	OK
20.	Menekan tombol Build.	Menampilkan gambar bangunan untuk di bangun.	Menampilkan gambar bangunan untuk di bangun.	OK

Berdasarkan hasil dari pengujian fungi Panel *Play Game* pada tabel 4.3 dapat di ambil kesimpulan bahwa semua hasil yang di harapkan sesuai tujuan.

3. Panel *Library*

Tabel 4.4
Pengujian Panel *Library*

No.	Pengujian	Tujuan	Hasil Diharapkan	Hasil Uji
1.	Menampilkan Data *Library* Jika sudah di akses.	Menampilkan Data *Cells* yang sudah di akses.	Menampilkan Data *Cells* yang sudah di akses.	OK
2.	Menekan tombol *Exit*.	Menutup panel *Library*.	Menutup panel *Library*.	OK

Berdasarkan hasil dari pengujian fungi Panel *Library* pada tabel 4.4 dapat di ambil kesimpulan bahwa semua hasil yang di harapkan sesuai tujuan.

4. Panel *Scores*

Tabel 4.5
Pengujian Panel *Scores*

No.	Pengujian	Tujuan	Hasil Diharapkan	Hasil Uji
1.	Menampilkan Data *Scores* yang sudah di save.	Menampilkan *Scores* yang sudah di raih oleh pemain.	Menampilkan *Scores* yang sudah di raih oleh pemain.	OK
2.	Menekan tombol *Exit*.	Menutup panel *Scores*.	Menutup panel *Scores*.	OK

Berdasarkan hasil dari pengujian fungi Panel *Scores* pada tabel 4.5 dapat di ambil kesimpulan bahwa semua hasil yang di harapkan sesuai tujuan.

5. Panel *Setting*

Tabel 4.6
Pengujian Panel *Setting*

No.	Pengujian	Tujuan	Hasil Diharapkan	Hasil Uji
1.	Menggerakan *Slider Music*.	Mengubah *Volume Music*.	Mengubah *Volume Music*.	OK
2.	Menggerakan *Slider Sound*.	Mengubah *Volume Sound*.	Mengubah *Volume Sound*.	OK
3.	Menekan tombol *Apply*.	Save data *setting Volume Music* & *Sound*.	Save data *setting Volume Music* & *Sound*.	OK
4.	Menekan tombol *Exit*.	Menutup panel *Setting*.	Menutup panel *Setting*.	OK

Berdasarkan hasil dari pengujian fungi Panel *Setting* pada tabel 4.6 dapat di ambil kesimpulan bahwa semua hasil yang di harapkan sesuai tujuan.

6. Panel *Menu Paused*

Tabel 4.7
Pengujian Panel *Menu Paused*

No.	Pengujian	Tujuan	Hasil Diharapkan	Hasil Uji
1.	Menggerakan *Slider Music*.	Mengubah *Volume Music*.	Mengubah *Volume Music*.	OK
2.	Menggerakan *Slider Sound*.	Mengubah *Volume Sound*.	Mengubah *Volume Sound*.	OK
3.	Menekan tombol *Apply*.	Save Data *Setting Volume Music* & *Sound*.	Save Data *Setting Volume Music* & *Sound*.	OK
4.	Menekan tombol *Resume.*	Melanjutkan *Game*.	Melanjutkan *Game*.	OK
5.	Menekan tombol *Exit*.	Menutup panel *Setting*.	Menutup panel *Setting*.	OK

Berdasarkan hasil dari pengujian fungi Panel *Menu Pause* pada tabel 4.7 dapat di ambil kesimpulan bahwa semua hasil yang di harapkan sesuai tujuan.

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dalam hasil pembuatan *game* dan pengujian "*Cells*" menggunakan Unity dan menggunakan pemograman C# untuk membuat *game real time strategy*, karena membutuhkan navigasi objek, pergerakan ai yang sederhana, dapat di ambil kesimpulan:

1. Telah memberi pengetahuan tentang *cell cell* tubuh manusia, bakteri, virus, dan parasite di *Library*. Menggambarkan peta yang menyamai dengan anatomi tubuh manusia menggunakan unity 3D.
2. *GameObject* digunakan sebagai aktor di dalam *game*.
3. AI *NavMesh* yang ada di dalam Unity untuk navigasi pergerakan *unit*.
4. *Collider* digunakan untuk mendeteksi musuh yang berada di dalam *collider*.
5. *Animator* digunakan untuk menggerakan animasi karakter 3D.
6. *SpriteRenderer* digunakan untuk menampilkan gambar 3D dari karakter.
7. *Camera* digunakan untuk melihat dalam *game*.
8. *Canvas* digunakan untuk menggambarkan antarmuka pemain.
9. *TextMeshPro* digunakan untuk menampilkan text.
10. *BinaryFormatter* digunakan untuk *save* data seperti data *level*, *scores,* dan *library.*

5.2. Saran

Game Cells masih dapat dikembangkan lebih bagus lagi, untuk memberi pengalaman bermain yang lebih menyenangkan dan informatif. Sebagian perkembangan yang dapat di lakukan seperti:

1. Memperbagus gambaran *terrain* seperti pegunungan, dan memperbanyak *terrain*.
2. Menambahkan interaksi terhadap minimap supaya pemain dapat menekan di lokasi *minimap* dan menggerakan kamera ke lokasi target *minimap*.
3. Pengaturan animasi yang kurang bagus dari script.
4. Memberikan fitur *multiplayer* untuk bermain bersama teman.
5. Menambahkan musuh fungi yang merubah *terrain* menjadi jamuran.
6. Menambahkan tipe musuh.

Daftar Pustaka

Chandler, Maxwell Chandler, 2011 "*Fundamentals of Game Development*", Jones & Bartlett Learning.

Clark, Dan, 2011, "*Beginning C# ObjectOriented Programming*", Apress.

Gede Wahyu, Ida Bagus, 2018, "Penerapan Algoritma A* (Star) Menggunakan Graph Untuk Menghitung Jarak Terpendek", Jurnal RESISTOR.

Nakov, Svetlin, Kolev, Veselin, 2013, "*Fundamentals of Computer Programming with C#*", Faber.

Pressman, Bruce, 2014, "*Software Engineering A Practitioner's Approach, 8th Edition*", McGraw-Hill Education.

Riskas, Georgios,et al, 2017, "*Macromanagement In RTS Game*", xamk.

Roger, Scott, 2014, "*Level UP! The Guide To Great Video Game Design 2nd Edition*", John Wiley & Sons.

Schmidt, Richard F, 2013, "*Software Engineering Architecture-Driven Software Development*", Elsevier Science.

Sommerville, Ian, 2011, "*Software Engineering, 9th Edition*", Pearson.

Sompayrac, LaurenJohn M, 2015, "*How the Immune System Works*", Wiley & Sons.

Wazlawick, Raul, 2014, "*Object-Oriented Analysis and Design for Information System*", Elsevier.

Wiles, 2017, "*Antibiotic Resistance The End of Modern Medicine*?", Bridget Williams Books

https://docs.unity3d.com/Manual/nav-NavigationSystem.html, diakses tanggal 9 April 2019, 10:35 WIB.

https://unity3d.com/unity, diakses tanggal 14 November 2018, jam 21.45 WIB.

https://www.uml-diagrams.org/uml-25-diagrams.html, diakses tanggal 23 November 2018, 2:14 WIB.

http://www.hematology.org/Patients/Basics/, diakses tanggal 23 November 2018, jam 2:32 WIB.

https://www.ucsfbenioffchildrens.org/education/what_is_bone_marrow/, diakses tanggal 29 November 2020, jam 17:12 WIB

https://medlineplus.gov/ency/patientinstructions/000104.htm, diakses tanggal 29 November 2020, jam 17:30 WIB

https://www.livescience.com/22665-nervous-system.html, diakses tanggal 29 November 2020, jam 17:40 WIB

https://www.webmd.com/, diakses tanggal 29 November 2020, jam 17:50 WIB

LAMPIRAN

LISTING PROGRAMING

1. *PlayerControls*

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.UI;
using UnityEngine.EventSystems;

public class PlayerControls : MonoBehaviour
{
    public List<GameObject> SelectedUnits = new List<GameObject>();
    public string playerCommand = "None";
    public GlobalAttribute MapGlobal;
    public GameObject MainUI;
    public bool gamePause = false;
    public bool cameraDisable = false;

    private bool clicked = false, drag = false;
    public Texture selectBoxTextue;
    private Vector3 dragMouse = Vector3.zero;
    private Vector2 boxStart = Vector2.zero;
    private Vector2 boxEnd = Vector2.zero;
    private Vector3 startPos, endPos;

    public List<GameObject> groupSelected1;
    public List<GameObject> groupSelected2;
    public List<GameObject> groupSelected3;
    public List<GameObject> groupSelected4;
    public List<GameObject> groupSelected5;
    ////////////////////////////////////////////Update////////////////////////////////////////
    private void Awake()
    {
        buttonBuilds[0] = GameObject.Find("btnBuild1").GetComponent<UIBuildTrainButton>();
        buttonBuilds[1] = GameObject.Find("btnBuild2").GetComponent<UIBuildTrainButton>();
        buttonBuilds[2] = GameObject.Find("btnBuild3").GetComponent<UIBuildTrainButton>();
        buttonBuilds[3] = GameObject.Find("btnBuild4").GetComponent<UIBuildTrainButton>();
        buttonBuilds[4] = GameObject.Find("btnBuild5").GetComponent<UIBuildTrainButton>();
        buttonBuilds[5] = GameObject.Find("btnBuild6").GetComponent<UIBuildTrainButton>();
        buttonBuilds[6] = GameObject.Find("btnBuild7").GetComponent<UIBuildTrainButton>();
    }
    void Update()
    {
        if (gamePause == false)
        {
            MousePress();
            KeyboardPress();

            BuildCommand();
        }
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            Pause();
        }
        //Update Info UI
        GameObject.Find("UIMain").GetComponent<UIScripts>().UIHealth(SelectedUnits);

    }
    ////////////////////////////////////////////Keyboard////////////////////////////////////////
    private bool ctrlHealthBar = true;
    void KeyboardPress()
    {
        if (SelectedUnits.Count > 0)
        {
            if (Input.GetKeyDown(KeyCode.S))
            {
                playerCommand = "None";
                changeStatusSelectedUnits("Stop");
            }
            else if (Input.GetKeyDown(KeyCode.A))
            {
                playerCommand = "Attack";
            }
```

```
        else if (Input.GetKeyDown(KeyCode.H))
        {
            changeStatusSelectedUnits("Hold");
        }
        else if (Input.GetKeyDown(KeyCode.M))
        {
            playerCommand = "Move";
        }

    }
    //Asign Group Units
    if (Input.GetKeyDown(KeyCode.Alpha1))
    {
        asignGroup(groupSelected1);
    }
    else if(Input.GetKeyDown(KeyCode.Alpha2))
    {
        asignGroup(groupSelected2);
    }
    else if (Input.GetKeyDown(KeyCode.Alpha3))
    {
        asignGroup(groupSelected3);
    }
    else if (Input.GetKeyDown(KeyCode.Alpha4))
    {
        asignGroup(groupSelected4);
    }
    else if (Input.GetKeyDown(KeyCode.Alpha5))
    {
        asignGroup(groupSelected5);
    }
    //Show Hide HealthBars (ctrlHealthBar)
    if (Input.GetKeyDown(KeyCode.LeftControl))
    {
        if (ctrlHealthBar)
        {
            Camera.main.cullingMask = (1 << LayerMask.NameToLayer("Default"))
            | (1 << LayerMask.NameToLayer("UI"))
            | (0 << LayerMask.NameToLayer("HPBar"));
            ctrlHealthBar = false;
        }
        else
        {
            Camera.main.cullingMask = (1 << LayerMask.NameToLayer("Default"))
              | (1 << LayerMask.NameToLayer("UI"))
              | (1 << LayerMask.NameToLayer("HPBar"));
            ctrlHealthBar = true;
        }
    }
}

public void Pause()
{
    if (gamePause)
    {
        Time.timeScale = 1;
    }
    else
    {
        Time.timeScale = 0;
        GameObject.Find("UIPauseMenu").GetComponent<SettingScript>().loadSetting();
    }
    GameObject.Find("UIPauseMenu").GetComponent<Canvas>().enabled = !gamePause;
    gamePause = !gamePause;
}

void asignGroup(List<GameObject> group)
{
    foreach (GameObject obj in SelectedUnits)
    {
        selectCircle(obj, false);
    }
```

```
        for (int i = 0; i < group.Count - 1; i++)
        {
            if (group[i] == null)
            {
                group.RemoveAt(i);
            }
        }
        if (Input.GetKey(KeyCode.LeftAlt))
        {
            Debug.Log("Group " + group);
            if (SelectedUnits.Count > 0)
            {
                group.Clear();
                foreach (GameObject obj in SelectedUnits)
                {
                    group.Add(obj);
                    selectCircle(obj, true);
                }
                GameObject.Find("UIMain").GetComponent<UIScripts>().UIInfo(SelectedUnits);
            }
        }
        else
        {
            if (group.Count > 0)
            {
                SelectedUnits.Clear();
                foreach (GameObject obj in group)
                {
                    SelectedUnits.Add(obj);
                    selectCircle(obj, true);
                }
                foreach (GameObject obj in SelectedUnits)
                {
                    selectCircle(obj, true);
                }
                GameObject.Find("UIMain").GetComponent<UIScripts>().UIInfo(SelectedUnits);
            }
        }

    }

    ////////////////////////////////////////Mouse////////////////////////////////////////
    void MousePress()
    {
        if (SelectedUnits.Count > 0)
        {
            if (Input.GetMouseButtonDown(1) && InsideUI() && tmpBuild == null)
            {
                RightClick();
            }
            //Build
            if (Input.GetMouseButtonDown(0) && InsideUI() && tmpBuild != null)
            {
                if (tmpBuild.GetComponent<SpriteDestroy>().buildpass)
                {
                    if (!place)
                    {
                        BuilderCancel(SelectedUnits[0]);
                        place = true;
                        tmpBuild.GetComponent<SpriteDestroy>().DestroyMe();
                        //Use Resource
                        gameObject.GetComponent<ResourceManager>().sugar -=
SelectedBuild.GetComponent<BuildingStatus>().sugarReq;
                        gameObject.GetComponent<ResourceManager>().oxygen -=
SelectedBuild.GetComponent<BuildingStatus>().sugarReq;
                        GameObject tmpi = Instantiate(SelectedBuild, new Vector3(mousePos().x, 0.5f,
mousePos().z), Quaternion.identity);
                        tmpi.transform.GetChild(3).GetComponent<BoxCollider>().enabled = false;
                        tmpi.GetComponent<BuildingStatus>().BuilderInside = SelectedUnits[0];
                        MoveSingle(0, tmpi);
                        removeIndexFromSelectedUnits(0);
                        playerCommand = "None";
                    }
```

```
            }
            else
            {
                SystemNotifText.Create("You Can Not Build There!",true , Color.red, 1.5f);
                place = true;
                tmpBuild.GetComponent<SpriteDestroy>().DestroyMe();
                playerCommand = "None";
            }
        }
    }
    if (Input.GetMouseButtonDown(0) && InsideUI() && tmpBuild == null)
    {
        startPos = mousePos();
        clicked = true;
        dragMouse = Input.mousePosition;
        boxStart = Input.mousePosition;
    }
    else if (Input.GetMouseButtonUp(0) && InsideUI() && tmpBuild == null)
    {
        LeftClick();
    }

    BecomeDrag();
    BuildListUpdate();
}


private void LeftClick()
{
    RaycastHit hitInfo = new RaycastHit();
    bool hit = Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hitInfo);
    if (hit)
    {
        if (playerCommand.Equals("None"))
        {
            if (hitInfo.transform.gameObject.tag == "PlayerSelectable")
            {
                selectObject(hitInfo);
            }
        }
        else if (playerCommand.Equals("Attack"))
        {
            if (hitInfo.transform.gameObject.tag == "EnemySelectable")
            {
                AttackTarget(hitInfo);
            }
            else
            {
                AttackMove(hitInfo);
            }
        }
        else if (playerCommand.Equals("Move"))
        {
            Move(hitInfo);
            playerCommand = "None";
        }
    }
}
private void RightClick()
{
    RaycastHit hitInfo = new RaycastHit();
    bool hit = Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hitInfo);
    if (hit)
    {
        if (hitInfo.transform.gameObject.tag == "EnemySelectable")
        {
            AttackTarget(hitInfo);
        }
        else
        {
            Move(hitInfo);
        }
    }
```

```
}


Vector3 mousePos()
{
    Vector3 spos = Vector3.zero;
    RaycastHit hitInfo = new RaycastHit();
    bool hit = Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hitInfo);
    if (hit)
    {
        spos = hitInfo.point;
    }
    return spos;
}

bool InsideUI()
{
    return !EventSystem.current.IsPointerOverGameObject();
}
////////////////////////////////////////Select box//////////////////////////////////////
void BecomeDrag()
{
    if (clicked == true && drag == false &&
          (Input.mousePosition.x > dragMouse.x + 0.8f ||
           Input.mousePosition.x < dragMouse.x - 0.8f ||
           Input.mousePosition.z > dragMouse.z + 0.8f ||
           Input.mousePosition.z < dragMouse.z - 0.8f))
    {
        drag = true;
        cameraDisable = true;
    }
    //Render the Select Box
    if (clicked == true && drag == true)
    {
        if (Input.GetKey(KeyCode.Mouse0))
        {
            if (Input.GetKeyDown(KeyCode.Mouse0))
                boxStart = Input.mousePosition;
            else
                boxEnd = Input.mousePosition;
        }
        else
        {
            if (boxEnd != Vector2.zero && boxStart != Vector2.zero)
                boxEnd = boxStart = Vector2.zero;
        }
    }
    if (Input.GetMouseButtonUp(0) && clicked == true)
    {
        endPos = mousePos();
        clicked = false;
        cameraDisable = false;
        if (drag == true)
        {
            findUnitInsideBox();
            drag = false;
        }
    }
}
void OnGUI()
{
    if (gamePause == false)
    {
        // Draw
        if (boxStart != Vector2.zero && boxEnd != Vector2.zero)
        {
            // Create a rectangle object out of the start and end position while transforming it
            // to the screen's cordinates.
            var rect = new Rect(boxStart.x, Screen.height - boxStart.y,
                                boxEnd.x - boxStart.x,
                                -1 * (boxEnd.y - boxStart.y));
            // Draw the texture.
            GUI.depth = -11;
```

```
            GUI.DrawTexture(rect, selectBoxTextue);
        }
    }
}
void findUnitInsideBox()
{
    GameObject[] units;
    units = GameObject.FindGameObjectsWithTag("PlayerSelectable");
    if (!Input.GetKey(KeyCode.LeftShift))
        deselectAll();
    int unitSelect = 0;
    foreach (GameObject unit in units)
    {
        if (ifInsideBox(unit))
        {
            if (unit.GetComponent<UnitStatus>() != null)
            {
                if (unit.GetComponent<UnitStatus>().enabled)
                {
                    SelectedUnits.Add(unit);
                    selectCircle(unit, true);
                    unitSelect++;
                }
            }
        }
    }
    if (unitSelect == 0)
    {
        bool first = false;
        string firstN = "";
        foreach (GameObject unit in units)
        {
            if (ifInsideBox(unit))
            {
                if (unit.GetComponent<BuildingStatus>() != null)
                {
                    if (unit.GetComponent<BuildingStatus>().enabled)
                    {
                        if (!first)
                        {
                            first = true;
                            firstN = unit.GetComponent<BuildingStatus>().name;
                        }
                        if(unit.GetComponent<BuildingStatus>().name.Equals(firstN))
                        {
                            SelectedUnits.Add(unit);
                            selectCircle(unit, true);
                        }
                    }

                }
            }
        }
    }

    GameObject.Find("UIMain").GetComponent<UIScripts>().UIInfo(SelectedUnits);

}
bool ifInsideBox(GameObject unit)
{
    bool tmp = false;
    if ((startPos.x < endPos.x && startPos.z < endPos.z &&
         unit.transform.position.x >= startPos.x && unit.transform.position.x <= endPos.x &&
         unit.transform.position.z >= startPos.z && unit.transform.position.z + 1 <= endPos.z) ||
          (startPos.x < endPos.x && startPos.z > endPos.z &&
         unit.transform.position.x >= startPos.x && unit.transform.position.x <= endPos.x &&
         unit.transform.position.z <= startPos.z && unit.transform.position.z + 1 >= endPos.z) ||
          (startPos.x > endPos.x && startPos.z < endPos.z &&
         unit.transform.position.x <= startPos.x && unit.transform.position.x >= endPos.x &&
         unit.transform.position.z + 1 >= startPos.z && unit.transform.position.z <= endPos.z) ||
          (startPos.x > endPos.x && startPos.z > endPos.z &&
         unit.transform.position.x <= startPos.x && unit.transform.position.x >= endPos.x &&
         unit.transform.position.z + 1 <= startPos.z && unit.transform.position.z >= endPos.z)
```

```
        )
    {
        tmp = true;
    }
    return tmp;
}
//////////////////////////////////////////Group//////////////////////////////////////////
public void deselectAll()
{
    if (SelectedUnits.Count != 0)
    {
        foreach (GameObject obj in SelectedUnits)
        {

            selectCircle(obj, false);
        }
        SelectedUnits.Clear();
        GameObject.Find("UIMain").GetComponent<UIScripts>().UIHealth(SelectedUnits);
    }
}


public void selectObject(RaycastHit hit)
{
    GameObject selectedObject = hit.transform.gameObject;
    if(!Input.GetKey(KeyCode.LeftShift))
        deselectAll();
    if (!SelectedUnits.Contains(selectedObject))
    {
        if (selectedObject.GetComponent<UnitStatus>() != null)
        {
            SelectedUnits.Add(selectedObject);
            selectCircle(selectedObject, true);
            //Debug.Log("Add(" + selectedObject.transform.GetChild(1).name + ")");
            GameObject.Find("UIMain").GetComponent<UIScripts>().UIInfo(SelectedUnits);
        }
        else if (selectedObject.GetComponent<BuildingStatus>() != null)
        {
            SelectedUnits.Add(selectedObject);
            selectCircle(selectedObject, true);
            //Debug.Log("Add(" + selectedObject.transform.GetChild(1).name + ")");
            GameObject.Find("UIMain").GetComponent<UIScripts>().UIInfo(SelectedUnits);
        }
    }
    else
    {
        SelectedUnits.Remove(selectedObject);
        selectCircle(selectedObject, false);
    }
}


public void removeObjectFromSelectedUnits(GameObject x)
{
    if (SelectedUnits.Contains(x))
    {
        SelectedUnits.Remove(x);
        GameObject.Find("UIMain").GetComponent<UIScripts>().UIHealth(SelectedUnits);
    }
}
public void removeIndexFromSelectedUnits(int x)
{
    if (SelectedUnits.Count - 1 > x)
        if (SelectedUnits.Contains(SelectedUnits[x]))
        {
            SelectedUnits.Remove(SelectedUnits[x]);
            GameObject.Find("UIMain").GetComponent<UIScripts>().UIHealth(SelectedUnits);
        }
}



Vector3 dynamicPosition(Vector3 mousePoint, int numberOfObject)
```

```
{
    Vector3 point = Vector3.zero;
    for (int i = 0; i < numberOfObject; i++)
    {
        point = TransformPointRangeAngle(mousePoint, 2, 120 * i);
        //Debug.Log(360 / numberOfObject * i);
    }

    return point;
}

Vector3 TransformPointRangeAngle(Vector3 startPoint, float distance, float angle)
{
    Transform poi = new GameObject().transform;
    poi.position = startPoint;
    poi.localRotation = Quaternion.Euler(0, angle, 0);
    poi.Translate(0, 0, distance);
    Vector3 endPoint = poi.position;
    Destroy(poi.gameObject);
    return endPoint;
}



//Unit commands
public void changeStatusSelectedUnits(string Command)
{
    foreach (GameObject obj in SelectedUnits)
    {
        if (Command.Equals("Hold") || Command.Equals("Stop"))
        {
            obj.GetComponent<NavMeshAgent>().destination = obj.transform.position;
        }
        BuilderCancel(obj);
        obj.GetComponent<UnitStatus>().State = Command;
        obj.GetComponent<UnitStatus>().Target = null;
        obj.GetComponent<UnitStatus>().flipReset = true;
    }
}

void AttackTarget(RaycastHit hitInfo)
{
    playerCommand = "None";
    foreach (GameObject obj in SelectedUnits)
    {
        BuilderCancel(obj);
        obj.GetComponent<NavMeshAgent>().destination = hitInfo.transform.position;
        obj.GetComponent<UnitStatus>().State = "AttackTarget";
        obj.GetComponent<UnitStatus>().Target = hitInfo.transform.gameObject;
        obj.GetComponent<UnitStatus>().flipReset = true;
    }
}

void Move(RaycastHit hitInfo)
{
    Vector3 pointRay;
    pointRay = hitInfo.point;
    MouseFade.Create(pointRay);
    int i = 0;
    foreach (GameObject obj in SelectedUnits)
    {
        BuilderCancel(obj);
        if (obj.GetComponent<UnitStatus>() != null)
        {
            obj.GetComponent<NavMeshAgent>().destination = pointRay;
            obj.GetComponent<UnitStatus>().State = "Move";
            obj.GetComponent<UnitStatus>().Target = null;
            obj.GetComponent<UnitStatus>().checkStates = false;
            obj.GetComponent<UnitStatus>().movePoint = pointRay;
            obj.GetComponent<UnitStatus>().flipReset = true;
            if (SelectedUnits.Count > 1)
```

```
                pointRay = TransformPointRangeAngle(hitInfo.point, 5, 180);//(360 / (SelectedUnits.Count - 1))
* i);
            }
            i++;
        }
        //Debug.Log(CalculateAngle(SelectedUnits[0].transform.position, hitInfo.point));
    }

    public void MoveSingle(int i, GameObject t)
    {
        SelectedUnits[i].GetComponent<NavMeshAgent>().destination = t.transform.position;
        SelectedUnits[i].GetComponent<UnitStatus>().State = "Move";
        SelectedUnits[i].GetComponent<UnitStatus>().Target = t;
        SelectedUnits[i].GetComponent<UnitStatus>().checkStates = false;
        SelectedUnits[i].GetComponent<UnitStatus>().movePoint = t.transform.position;
        SelectedUnits[i].GetComponent<UnitStatus>().flipReset = true;
    }

    void AttackMove(RaycastHit hitInfo)
    {
        playerCommand = "None";
        Vector3 pointRay;
        pointRay = hitInfo.point;
        int i = 0;
        foreach (GameObject obj in SelectedUnits)
        {
            BuilderCancel(obj);
            i++;
            obj.GetComponent<NavMeshAgent>().destination = pointRay;
            obj.GetComponent<UnitStatus>().State = "AttackMove";
            obj.GetComponent<UnitStatus>().Target = null;
            obj.GetComponent<UnitStatus>().flipReset = true;
            pointRay = dynamicPosition(hitInfo.point, i);
        }
    }


    void BuilderCancel(GameObject o)
    {
        if (o.GetComponent<UnitStatus>() != null && o.GetComponent<UnitStatus>().Target != null)
        {
            if (o.GetComponent<UnitStatus>().Target.GetComponent<BuildingStatus>() != null)
            {
                if (o.GetComponent<UnitStatus>().isBuilder)
                {
                    GameObject b = o.GetComponent<UnitStatus>().Target;
                    //Refund
                    gameObject.GetComponent<ResourceManager>().sugar +=
SelectedBuild.GetComponent<BuildingStatus>().sugarReq;
                    gameObject.GetComponent<ResourceManager>().oxygen +=
SelectedBuild.GetComponent<BuildingStatus>().sugarReq;
                    b.GetComponent<BuildingStatus>().DestroyMe();
                }
            }
        }
    }

    void selectCircle(GameObject obj, bool b)
    {
        if (obj.GetComponent<UnitStatus>() != null)
            obj.transform.GetChild(1).GetComponent<SpriteRenderer>().enabled = b;
        else if (obj.GetComponent<BuildingStatus>() != null)
            obj.transform.GetChild(2).GetComponent<SpriteRenderer>().enabled = b;
    }

    ////////////////////////////////////////PlayerBuildCommand////////////////////////////////////////
    public GameObject SelectedBuild;
    public List<GameObject> BuildList;


    private bool place = true;
    public GameObject tmpBuild;
```

```
    private bool buildOnUI;
    void BuildCommand()
    {
        if (!place && tmpBuild !=null)
        {
            if (InsideUI())
            {
                ProjectingBuild();
            }
            if (Input.GetMouseButtonDown(1))
            {
                place = true;
                tmpBuild.GetComponent<SpriteDestroy>().DestroyMe();
                playerCommand = "None";
            }
        }

    }

    public void ProjectingBuild()
    {
        tmpBuild.GetComponent<Transform>().position = new Vector3(mousePos().x, 0.5f, mousePos().z);
    }
    public void createProjecting(GameObject obj)
    {
        if (obj != null)
        {
            //BuildList = ScriptPlayerMoveCommand.SelectedUnits[0].GetComponent<UnitStatus>().BuildList;
            SelectedBuild = obj;
            tmpBuild = Instantiate(SelectedBuild.transform.GetChild(3).gameObject, new Vector3(0, 0.5f, 0),
Quaternion.identity);
            tmpBuild.transform.GetChild(0).GetComponent<SpriteRenderer>().enabled = true;
            foreach (Transform child in tmpBuild.transform)
            {
                if (!child.name.Equals("BuildingBoarder"))
                {
                    child.GetComponent<SpriteRenderer>().color = new Color32(255, 255, 255, 100);
                }
            }
            //= new Color32(255, 255, 255, 100)

            place = false;
        }
    }
    private UIBuildTrainButton[] buttonBuilds = new UIBuildTrainButton[7];
    void BuildListUpdate()
    {
        if (SelectedUnits.Count > 0)
        {
            if (SelectedUnits[0].GetComponent<UnitStatus>() != null)
                if (SelectedUnits[0].GetComponent<UnitStatus>().BuildList.Count > 0)
                {
                    BuildList = SelectedUnits[0].GetComponent<UnitStatus>().BuildList;

                    for (int i = 0; i < 7; i++)
                    {
                        if (i < SelectedUnits[0].GetComponent<UnitStatus>().BuildList.Count)
                            buttonBuilds[i].statusGameObject =
SelectedUnits[0].GetComponent<UnitStatus>().BuildList[i];
                        else
                            buttonBuilds[i].statusGameObject = null;
                        buttonBuilds[i].readGameObject();
                    }
                }
        }
        else
        {
            for (int i = 0; i < 7; i++)
            {
                if(buttonBuilds[i].statusGameObject != null)
                {
                    buttonBuilds[i].statusGameObject = null;
                    buttonBuilds[i].readGameObject();
```

```
            }
        }
    }


}

public void buildButton(int i)
{
    ResourceManager RM = gameObject.GetComponent<ResourceManager>();
    Debug.Log(BuildList[i].name);
    if (BuildList[i].GetComponent<UnitStatus>() != null)
    {
        if (RM.sugar >= BuildList[i].GetComponent<UnitStatus>().sugarReq &&
            RM.oxygen >= BuildList[i].GetComponent<UnitStatus>().oxygenReq)
        {
            if (RM.maxcells - RM.cells > BuildList[i].GetComponent<UnitStatus>().cellsReq)
            {
                //Train Unit
            }
            else
            {
                //CellTooFull
                SystemNotifText.Create("Max Cells!", true, Color.red, 1.5f);
            }
        }
        else
        {
            //NotEnoughResource
            SystemNotifText.Create("Not Enough Resources!", true, Color.red, 1.5f);
        }
    }
    else if (BuildList[i].GetComponent<BuildingStatus>() != null)
    {
        if (RM.sugar >= BuildList[i].GetComponent<BuildingStatus>().sugarReq &&
            RM.oxygen >= BuildList[i].GetComponent<BuildingStatus>().oxygenReq)
        {
            if (RM.maxcells - RM.cells > BuildList[i].GetComponent<BuildingStatus>().cellsReq)
            {
                //BUILD COMMAND
                if (tmpBuild != null)
                {
                    tmpBuild.GetComponent<SpriteDestroy>().DestroyMe();
                    tmpBuild = null;
                }
                if (SelectedUnits[0].GetComponent<UnitStatus>() != null)
                {
                    if (SelectedUnits[0].GetComponent<UnitStatus>().isBuilder)
                    {
                        playerCommand = "Build";
                        //BuildList =
ScriptPlayerMoveCommand.SelectedUnits[0].GetComponent<UnitStatus>().BuildList;
                        createProjecting(BuildList[i]); //Change Base on the index build
                        showHideBuildingBoarder(true);
                    }
                }
            }
            else
            {
                //CellTooFull
                SystemNotifText.Create("Max Cells!", true, Color.red, 1.5f);
            }
        }
        else
        {
            //NotEnoughResource
            SystemNotifText.Create("Not Enough Resources!", true, Color.red, 1.5f);
        }
    }
}

public static void showHideBuildingBoarder(bool t)
{
```

```
            GameObject[] tmp = GameObject.FindGameObjectsWithTag("PlayerSelectable");
            foreach (GameObject obj in tmp)
            {
                if (obj.GetComponent<BuildingStatus>() != null)
                {
                    obj.transform.GetChild(3).GetChild(0).GetComponent<SpriteRenderer>().enabled = t;
                }
            }
        }
    }
```

2. *UnitStatus*

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

public class UnitStatus : MonoBehaviour
{
    ////////////////////////////////////////Property/////////////////////////////////////////
    public int sugarReq = 0;
    public int oxygenReq = 0;
    public int cellsReq = 1;
    public string description = "";
    public int trainTime = 10;

    public string UnitName = "";
    public string State = "Stop"; //Stop, Attack, Move, AttackMove, Hold, AttackTarget, Attacking
    public int Health = 10;
    public int MaxHealth = 10;
    public GameObject Target = null;
    public Sprite missileSprite = null;
    public float missileTime = 0;
    public string DamageType = "None";
    public int Damage = 1;
    public int MaxDamage = 2;
    public float Range = 1.4f; //0-1.4f (Melee) max range 9
    public float attackSpeed = 3f;
    public float attackDelay = 0.35f;
    public string ArmourType = "None";
    public int Armour = 1;

    public bool enemyAIMove = false;

    public GlobalAttribute MapScript;
    public Sprite iconSprite;

    private string AI_MainTarget = "HQ";
    //
    private Animator anim;

    public Vector3 movePoint;

    public bool isBuilder = false;
    public List<GameObject> BuildList;

    private void Start()
    {
        MapScript = GameObject.Find("Map").GetComponent<GlobalAttribute>();
        foreach (Transform child in transform)
        {
            if (child.name == "Sprite")
            {
                anim = child.GetComponent<Animator>();
            }
        }
        if (gameObject.tag == "EnemySelectable")
        {

gameObject.transform.GetChild(0).transform.GetChild(0).gameObject.GetComponent<SpriteRenderer>().color
= Color.red;
```

```
            gameObject.transform.GetChild(3).gameObject.GetComponent<SpriteRenderer>().color = Color.red;
        }
    }

    ////////////////////////////////////////Animation Part (Need Rework)////////////////////////////////////////
    private float flipDelay = 0.1f;
    private int flipBuged = 0;
    public bool flipReset = false;
    private Vector3 pastPoint;
    void animations()
    {
        if (State.Equals("Move") || State.Equals("AttackMove") || State.Equals("AttackTarget"))
        {
            anim.SetBool("isWalking", true);
        }
        else
        {
            anim.SetBool("isWalking", false);
        }

        if (flipReset == true) { flipBuged = 0; flipReset = false; }
        flipDelay -= Time.deltaTime;
        if (flipDelay < 0)
        {
            pastPoint = gameObject.transform.position;
            flipDelay = 0.1f;
        }
        if(gameObject.transform.position.x < pastPoint.x && facingRight == true)
        {
            if (flipBuged < 7)
                Flip();
        }
        else if (gameObject.transform.position.x > pastPoint.x && facingRight == false)
        {
            if (flipBuged < 7)
                Flip();
        }
        gameObject.transform.GetChild(0).transform.GetChild(0).gameObject.transform.localScale = new
Vector3((float)Health / (float)MaxHealth, 1,1);
    }
    private bool facingRight = true;
    void Flip()
    {
        facingRight = !facingRight;
        Vector3 theScale = transform.localScale;
        theScale.x *= -1;
        transform.localScale = theScale;


        GameObject hpChild = gameObject.transform.GetChild(0).transform.GetChild(0).gameObject;
        hpChild.gameObject.transform.localPosition = new Vector3(hpChild.gameObject.transform.localPosition.x *
-1, 0, 0);
        hpChild.GetComponent<SpriteRenderer>().flipX = !facingRight;
        flipBuged++;
    }


    public bool checkStates = false;
    private Vector3 checkStopLocation;
    private float checkStopTimer = 5f;
    void AnimationStates()
    {
        if (gameObject.transform.position.x + 1 < checkStopLocation.x || gameObject.transform.position.x - 1 >
checkStopLocation.x)
        {
            checkStopLocation = gameObject.transform.position;
            checkStopTimer = 5f;
        }
        else
        {
            if (State == "Attack" || State == "Move" || State == "AttackMove")
            {
                if (checkStopTimer < 0)
```

```
            {
                checkStopLocation = gameObject.transform.position + new Vector3(2, 0, 2);
                State = "Stop";
                gameObject.GetComponent<NavMeshAgent>().destination = gameObject.transform.position;
            }
            else
            {
                checkStopTimer -= Time.deltaTime;
            }
        }
    }

    if (gameObject.transform.position.x == movePoint.x && gameObject.transform.position.z == movePoint.z
&& checkStates == false)
    {
        State = "Stop";
        checkStates = true;
    }
}

//////////////////////////////////////////Update//////////////////////////////////////////
public float addedRange = 0;
private void Update()
{
    if (anim != null)
    {
        animations();
    }
    AnimationStates();
    ///////////////

    if ((State.Equals("Attacking")|| State.Equals("AttackTarget")) && Target != null)
    {
        if (Target.GetComponent<BuildingStatus>() != null)
        {
            NavMeshObstacle obs = Target.GetComponent<NavMeshObstacle>();
            if (obs.size.x > obs.size.y)
                addedRange = obs.size.x + (2 / 2 - Range);
            else
                addedRange = obs.size.y + (2  / 2 - Range);
            if (Range > 2)
                addedRange += Range;
        }
        else
        {
            addedRange = 0;
        }
        if (Vector3.Distance(gameObject.transform.position, Target.gameObject.transform.position) < (Range +
addedRange) * 2 + (Range * 0.1f))
        {
            attacking();
        }
        else
        {
            //if(gameObject.tag == "EnemySelectable")
            //   Target = checkClosest();
            checkTarget();
        }
    }

    //IF States
    if ((State.Equals("AttackMove") || State.Equals("Stop")))
    {
        checkTarget();
    }
    else if ((State.Equals("Attacking") || State.Equals("AttackTarget")) && Target == null)
    {
        checkTarget();
    }

    //////////////
    isUnitDead();
}
```

```
    void isUnitDead()
    {
        if (Health <= 0)
        {
            if (gameObject.tag == "PlayerSelectable")

GameObject.Find("Player").GetComponent<PlayerControls>().removeObjectFromSelectedUnits(gameObject);
            Transform tmp = gameObject.transform.GetChild(2);
            tmp.parent = null;
            if (gameObject.tag == "PlayerSelectable")
                tmp.GetChild(0).GetChild(0).GetChild(0).GetComponent<Animator>().SetBool("isDeath", true);
            tmp.GetComponent<Animator>().SetBool("isDeath",true);
            tmp.GetComponent<SpriteDeath>().enabled = true;
            tmp.GetComponent<SpriteDeath>().death = true;
            tmp.GetComponent<SpriteDeath>().deathTime = 3;

            Destroy(gameObject);
        }
    }

    ////////////////////////////////////////Attacking////////////////////////////////////////
    public List<GameObject> listEnemy = new List<GameObject>();
    GameObject checkClosest()
    {
        GameObject Closest = null;
        float range = 100;
        for (int i = 0; i < listEnemy.Count; i++)
        {
            if (listEnemy[i].gameObject == null)
            {
                listEnemy.Remove(listEnemy[i].gameObject);
                break;
            }
            else
            {
                if (Vector3.Distance(gameObject.transform.position, listEnemy[i].gameObject.transform.position) <
range)
                {
                    if (checkScriptEnabled(listEnemy[i]))
                    {
                        Closest = listEnemy[i];
                        range = Vector3.Distance(gameObject.transform.position,
listEnemy[i].gameObject.transform.position);
                    }
                }
            }
            //Debug.Log("Enemy: " + listEnemy[cnt].name + " Distance: " +
Vector3.Distance(gameObject.transform.position, listEnemy[cnt].gameObject.transform.position));
        }
        if (Closest != null)
        {
            flipReset = true;
            Target = Closest;
            State = "AttackMove";
            gameObject.GetComponent<NavMeshAgent>().destination = Closest.transform.position;
        }
        return Closest;
        //Debug.Log("Closeset " + Closest.name);
    }
    private bool checkScriptEnabled(GameObject x)
    {
        bool tmp = false;
        if (x.GetComponent<UnitStatus>() != null)
            if (x.GetComponent<UnitStatus>().enabled)
                tmp = true;
        if (x.GetComponent<BuildingStatus>() != null)
            if (x.GetComponent<BuildingStatus>().enabled)
                tmp = true;
        return tmp;
    }
```

```
    private void checkTarget()
    {
        if (Target != null)
        {
            if (Target.GetComponent<BuildingStatus>() != null)
            {
                NavMeshObstacle obs = Target.GetComponent<NavMeshObstacle>();
                if (obs.size.x > obs.size.y)
                    addedRange = obs.size.x + (2 / 2 - Range);
                else
                    addedRange = obs.size.y + (2 / 2 - Range);
                if (Range > 2)
                    addedRange += Range;
                Debug.Log(obs.size.x);
            }
            else
            {
                addedRange = 0;
            }
            if (checkScriptEnabled(Target))
                if (Vector3.Distance(gameObject.transform.position, Target.gameObject.transform.position) < (Range
+ addedRange)*2)
                {
                    gameObject.GetComponent<NavMeshAgent>().destination = gameObject.transform.position;
                    State = "Attacking";
                    attackTimer = attackSpeed;
                    attackTimerDelay = attackDelay;
                    attackOnce = false;
                }
                else
                {
                    if (!State.Equals("AttackTarget"))
                    {
                        gameObject.GetComponent<NavMeshAgent>().destination = Target.transform.position;
                        State = "AttackMove";
                        attackTimer = attackSpeed;
                        attackTimerDelay = attackDelay;
                        attackOnce = false;
                        //AI(Enemy) if target null
                        Target = checkClosest();
                        if (gameObject.tag == "EnemySelectable" && Target == null)
                        {
                            gameObject.GetComponent<NavMeshAgent>().destination =
GameObject.Find(AI_MainTarget).transform.position;
                        }
                    }

                }
            else
            {
                Target = null;
            }
        }
        else
        {
            if(!isBuilder || State.Equals("AttackMove"))
            Target = checkClosest();
            if (enemyAIMove && Target == null && gameObject.tag == "EnemySelectable")
            {
                gameObject.GetComponent<NavMeshAgent>().destination =
GameObject.Find(AI_MainTarget).transform.position;
                State = "AttackMove";
                attackTimer = attackSpeed;
                attackTimerDelay = attackDelay;
                attackOnce = false;
            }
        }
    }


    public float attackTimer;
    public float attackTimerDelay;
    private bool attackOnce = false;
```

```
    void attacking()
    {
        if (checkScriptEnabled(Target))
        {
            if (attackTimer < 0)
            {
                if (!attackOnce)
                {
                    attackOnce = true;
                    anim.SetTrigger("Attack");
                }
                if (attackTimerDelay < 0)
                {
                    //Damages the target
                    if (missileSprite == null)
                    {
                        damageTarget(gameObject, Target);
                    }
                    else
                    {
                        MissileScript.Create(Target, calcDamageFromGlobal(),
TransformPointRangeAngle(TransformPointRangeAngle(gameObject.transform.position,1.6f,0), 0.7f,
                            CalculateAngle(gameObject.transform.position, Target.transform.position)),
                            missileTime, missileSprite);
                    }
                    attackTimer = attackSpeed;
                    attackTimerDelay = attackDelay;
                    attackOnce = false;
                }
                else
                {
                    attackTimerDelay -= Time.deltaTime;
                }
            }
            else
            {
                attackTimer -= Time.deltaTime;
            }
        }
        else
        {
            Target = null;
            attackTimer = attackSpeed;
            attackTimerDelay = attackDelay;
        }
        if (Target == null)
        {
            State = "Stop";
        }
    }

    void damageTarget(GameObject attackingUnit, GameObject attackedUnit)
    {
        int FinalDamage = 0;
        if (attackedUnit != null && State.Equals("Attacking"))
        {////////////////////////////////////////////////////////////////////////////////////
            if (attackedUnit.GetComponent<UnitStatus>() != null)
            {
                FinalDamage = calcDamageFromGlobal();
                attackedUnit.GetComponent<UnitStatus>().Health =
                attackedUnit.GetComponent<UnitStatus>().Health - FinalDamage;
                //If Builder Gets Damaged Run Away
                if (attackedUnit.GetComponent<UnitStatus>().isBuilder &&
attackedUnit.GetComponent<UnitStatus>().Target == null)
                {
                    Vector3 tmp = TransformPointRangeAngle(attackedUnit.transform.position, 4.5f,
CalculateAngle(gameObject.transform.position, attackedUnit.transform.position));
                    attackedUnit.GetComponent<UnitStatus>().State = "Move";
                    attackedUnit.GetComponent<UnitStatus>().Target = null;
                    attackedUnit.GetComponent<UnitStatus>().checkStates = false;
                    attackedUnit.GetComponent<UnitStatus>().flipReset = true;
                    attackedUnit.GetComponent<UnitStatus>().movePoint = tmp;
```

```
                attackedUnit.GetComponent<NavMeshAgent>().destination = tmp;
            }
        }
        else if (attackedUnit.GetComponent<BuildingStatus>() != null)
        {
            FinalDamage = calcDamageFromGlobal();
            attackedUnit.GetComponent<BuildingStatus>().Health =
            attackedUnit.GetComponent<BuildingStatus>().Health - FinalDamage;
        }
        if(gameObject.tag == "PlayerSelectable")
            DamagePopup.Create(attackedUnit.transform.position, "-", FinalDamage, 255, 0, 0, 255);
        else
            DamagePopup.Create(attackedUnit.transform.position, "-", FinalDamage, 0, 100, 0, 255);

    }
    else
    {
        State = "Stop";
    }
}

int calcDamageFromGlobal()
{
    int damage = 0;
    if (Target.GetComponent<UnitStatus>() != null)
    {
        damage = MapScript.damageCalculator(Random.Range(Damage, MaxDamage + 1), DamageType,
Target.GetComponent<UnitStatus>().ArmourType, Target.GetComponent<UnitStatus>().Armour);
    }
    else if (Target.GetComponent<BuildingStatus>() != null)
    {
        damage = MapScript.damageCalculator(Random.Range(Damage, MaxDamage + 1), DamageType,
Target.GetComponent<BuildingStatus>().ArmourType, Target.GetComponent<BuildingStatus>().Armour);
    }
    return damage;
}

//////////////////////////////////////////Update//////////////////////////////////////////
Vector3 TransformPointRangeAngle(Vector3 startPoint, float distance, float angle)
{
    Transform poi = new GameObject().transform;
    poi.position = startPoint;
    poi.localRotation = Quaternion.Euler(0, angle, 0);
    poi.Translate(0, 0, distance);
    Vector3 endPoint = poi.position;
    Destroy(poi.gameObject);
    return endPoint;
}
float CalculateAngle(Vector3 from, Vector3 to)
{
    return (Mathf.Atan2(from.z - to.z, to.x - from.x) * Mathf.Rad2Deg) + 90f;
}

//////////////////////////////////////////Builder Only//////////////////////////////////////////

//////////////////////////////////////////Collider//////////////////////////////////////////
private void OnTriggerEnter(Collider other)
{
    string tmp = "";
    if (gameObject.tag == "PlayerSelectable")
    {
        tmp = "EnemySelectable";
    }
    else if (gameObject.tag == "EnemySelectable")
    {
        tmp = "PlayerSelectable";
    }
    if (other.gameObject.tag == tmp && !other.isTrigger && !listEnemy.Contains(other.gameObject))
    {
        listEnemy.Add(other.gameObject);
    }
}
private void OnTriggerExit(Collider other)
```

```
    {
        listEnemy.Remove(other.gameObject);
    }
    ////////////////////////////////////////Hide////////////////////////////////////////
    public void HideMe(bool t)
    {
        if(gameObject != null)
        {
            State = "Stop";
            Target = null;
            anim.SetBool("isWalking", false);
            gameObject.GetComponent<NavMeshAgent>().destination = gameObject.transform.position;
            getallchild(gameObject.transform, t);
            if (gameObject.GetComponent<CapsuleCollider>() != null)
                gameObject.GetComponent<CapsuleCollider>().enabled = !t;
            if (gameObject.GetComponent<UnitStatus>() != null)
                gameObject.GetComponent<UnitStatus>().enabled = !t;
            if (gameObject.GetComponent<SphereCollider>() != null)
                gameObject.GetComponent<SphereCollider>().enabled = !t;
            gameObject.transform.GetChild(1).GetComponent<SpriteRenderer>().enabled = false;

        }
    }
    void getallchild(Transform p, bool t)
    {
        if (p.transform.childCount > 0)
        {
            foreach (Transform child in p)
            {
                if (child.GetComponent<Animator>() != null)
                    child.GetComponent<Animator>().enabled = !t;
                if (child.GetComponent<SpriteRenderer>() != null)
                    child.GetComponent<SpriteRenderer>().enabled = !t;
                getallchild(child, t);
            }
        }
    }
}
```

3. *BuildingStatus*

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

public class BuildingStatus : MonoBehaviour
{
    public int sugarReq = 0;
    public int oxygenReq = 0;
    public int cellsReq = 0;
    public string description = "";
    public int buildTime = 10;

    public string BuildingName = "Building";
    public string State = "WaitBuild"; //Stop, Building, Training, WaitBuild
    public int Health = 1;
    public int MaxHealth = 100;
    public string ArmourType = "Building";
    public int Armour = 1;

    public Sprite iconSprite;

    public Vector3 RallyPoint;
    public List<GameObject> listTraining = new List<GameObject>();
    public List<int> listTrainingTimer = new List<int>();
    public List<int> listQue = new List<int>();

    public GameObject BuilderInside; //Hide Building Unit when Constructing
    private void Start()
    {
        RallyPoint = gameObject.transform.position;
```

```
        if (buildTime > MaxHealth)
            buildTime = MaxHealth;

        if (gameObject.tag == "EnemySelectable")
        {

gameObject.transform.GetChild(0).transform.GetChild(0).gameObject.GetComponent<SpriteRenderer>().color
= Color.red;
            gameObject.transform.GetChild(4).gameObject.GetComponent<SpriteRenderer>().color = Color.red;
        }
        else if (gameObject.tag == "Neutral")
        {
            gameObject.transform.GetChild(4).gameObject.GetComponent<SpriteRenderer>().color = Color.grey;
        }
    }

    void isBuildingDead()
    {
        if (Health <= 0)
        {
            if (BuilderInside != null)
            {
                BuilderInside.GetComponent<UnitStatus>().HideMe(false);
            }
            if (gameObject.tag == "PlayerSelectable")

GameObject.Find("Player").GetComponent<PlayerControls>().removeObjectFromSelectedUnits(gameObject);
            Destroy(gameObject);
        }
    }

    private float timer;
    void isStartBuilding()
    {
        if (State.Equals("WaitBuild"))
        {
            if (BuilderInside != null)
            {
                if(Vector3.Distance(gameObject.transform.position, BuilderInside.gameObject.transform.position) <
3.4f) // Plus Scale of Building Other Wise it Wont Reach
                {

gameObject.transform.GetChild(0).transform.GetChild(0).gameObject.GetComponent<SpriteRenderer>().color
= Color.yellow;
                    BuilderInside.GetComponent<UnitStatus>().HideMe(true);
                    PlayerControls a = GameObject.Find("Player").GetComponent<PlayerControls>();
                    if (a != null)
                        a.removeObjectFromSelectedUnits(BuilderInside);
                    BuilderInside.GetComponent<NavMeshAgent>().destination = BuilderInside.transform.position;

                    BuilderInside.GetComponent<UnitStatus>().State = "Stop";
                    BuilderInside.GetComponent<UnitStatus>().Target = null;
                    BuilderInside.GetComponent<UnitStatus>().flipReset = true;
                    State = "Building";
                }

            }
            else
            {
                DestroyMe();
            }
        }
        if (State.Equals("Building"))
        {
            if (timer >= 1f)
            {
                Health = Health + (MaxHealth / buildTime);
                if (Health >= MaxHealth)
                {
                    if (BuilderInside != null)
                    {
                        BuilderInside.GetComponent<UnitStatus>().HideMe(false);
                    }
```

```
                BuilderInside = null;
                Health = MaxHealth;
                State = "Stop";

gameObject.transform.GetChild(0).transform.GetChild(0).gameObject.GetComponent<SpriteRenderer>().color
= Color.green;
            }
            timer = 0;
        }
        else
        {
            timer += Time.deltaTime;
        }
    }
}
private float trainTimer;
void isStartTraining()
{
    if (State.Equals("Training"))
    {
        if (listQue.Count <= 0)
        {
            State = "Stop";
        }
        else
        {
            if (timer >= 1f)
            {
                trainTimer++;
                gameObject.transform.GetChild(1).transform.GetChild(0).gameObject.transform.localScale =
                    new Vector3((float)trainTimer / (float)listTrainingTimer[listQue[0]], 1, 1);
                if (trainTimer >= listTrainingTimer[listQue[0]])
                {
                    Instantiate(listTraining[0], new Vector3(gameObject.transform.position.x + Random.Range(-
1f,1f), gameObject.transform.position.y, gameObject.transform.position.z -3 + Random.Range(-1f, 1f)),
Quaternion.identity);
                    //DamagePopup.Create(gameObject.transform.position, "-Training ", 1, 255, 0, 0, 255);
                    trainTimer = 0;
                    gameObject.transform.GetChild(1).transform.GetChild(0).gameObject.transform.localScale =
                        new Vector3((float)trainTimer / (float)listTrainingTimer[listQue[0]], 1, 1);
                    listQue.RemoveAt(0);
                }

                timer = 0;
            }
            else
            {
                timer += Time.deltaTime;
            }
        }
    }
}
void addQueFromList(int x)
{
    if(listQue.Count < 5)
    listQue.Add(x);
}
void Update()
{
    isStartBuilding();
    isStartTraining();
    isBuildingDead();
    gameObject.transform.GetChild(0).transform.GetChild(0).gameObject.transform.localScale = new
Vector3((float)Health / (float)MaxHealth, 1, 1);
}

public void DestroyMe()
{
    Destroy(gameObject);
}

}
```

4. *Player*

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

public class Player : MonoBehaviour
{
    public int level;
    public int settingMusic;
    public int settingSound;

    public int scoreExterminate;
    public int scoreDefend;
    public int scoreSurvival;

    //playerUnits
    public bool libraryPRedBloodCell;
    public bool libraryPTrombocytes;
    public bool libraryPWhiteBloodCell;
    public bool libraryPMacrophage;
    public bool libraryPBCell;
    public bool libraryPTCell;
    public bool libraryPEosinophils;

    //playerBuildings
    public bool libraryPNerveCenter;
    public bool libraryPIntestine;
    public bool libraryPArtery;
    public bool libraryPLymphNode;
    public bool libraryPBoneMarrow;
    public bool libraryPFat;
    public bool libraryPCellWall;

    //enemyUnits
    public bool libraryEStreptococcusPneumoniae;
    public bool libraryEStaphyllococcusAureus;
    public bool libraryEInfluenza;
    public bool libraryEAnisakis;


    private void Start()
    {
        LoadPlayer();
    }

    public void SavePlayer()
    {
        SaveScript.Save(this);
    }

    public void LoadPlayer()
    {
        PlayerData data = SaveScript.LoadPlayer();

        level = data.level;
        settingMusic = data.settingMusic;
        settingSound = data.settingSound;

        scoreExterminate = data.scoreExterminate;
        scoreDefend = data.scoreDefend;
        scoreSurvival = data.scoreSurvival;

        //playerUnits
        libraryPRedBloodCell = data.libraryPRedBloodCell;
        libraryPTrombocytes = data.libraryPTrombocytes;
        libraryPWhiteBloodCell = data.libraryPWhiteBloodCell;
        libraryPMacrophage = data.libraryPMacrophage;
        libraryPBCell = data.libraryPBCell;
        libraryPTCell = data.libraryPTCell;
        libraryPEosinophils = data.libraryPEosinophils;
```

```
        //playerBuildings
        libraryPNerveCenter = data.libraryPNerveCenter;
        libraryPIntestine = data.libraryPIntestine;
        libraryPArtery = data.libraryPArtery;
        libraryPLymphNode = data.libraryPLymphNode;
        libraryPBoneMarrow = data.libraryPBoneMarrow;
        libraryPFat = data.libraryPFat;
        libraryPCellWall = data.libraryPCellWall;

        //enemyUnits
        libraryEStreptococcusPneumoniae = data.libraryEStreptococcusPneumoniae;
        libraryEStaphyllococcusAureus = data.libraryEStaphyllococcusAureus;
        libraryEInfluenza = data.libraryEInfluenza;
        libraryEAnisakis = data.libraryEAnisakis;
    }

    public void ResetSave()
    {
        PlayerData data = SaveScript.LoadPlayer();

        level = 0;
        settingMusic = 100;
        settingSound = 100;

        scoreExterminate = 0;
        scoreDefend = 0;
        scoreSurvival = 0;

        //playerUnits
        libraryPRedBloodCell = false;
        libraryPTrombocytes = false;
        libraryPWhiteBloodCell = false;
        libraryPMacrophage = false;
        libraryPBCell = false;
        libraryPTCell = false;
        libraryPEosinophils = false;

        //playerBuildings
        libraryPNerveCenter = false;
        libraryPIntestine = false;
        libraryPArtery = false;
        libraryPLymphNode = false;
        libraryPBoneMarrow = false;
        libraryPFat = false;
        libraryPCellWall = false;

        //enemyUnits
        libraryEStreptococcusPneumoniae = false;
        libraryEStaphyllococcusAureus = false;
        libraryEInfluenza = false;
        libraryEAnisakis = false;

        SavePlayer();
    }
}
```

5. *PlayerData*

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

[System.Serializable]
public class PlayerData
{
    public int level;
    public int settingMusic;
    public int settingSound;

    public int scoreExterminate;
    public int scoreDefend;
    public int scoreSurvival;
```

```
    //playerUnits
    public bool libraryPRedBloodCell;
    public bool libraryPTrombocytes;
    public bool libraryPWhiteBloodCell;
    public bool libraryPMacrophage;
    public bool libraryPBCell;
    public bool libraryPTCell;
    public bool libraryPEosinophils;

    //playerBuildings
    public bool libraryPNerveCenter;
    public bool libraryPIntestine;
    public bool libraryPArtery;
    public bool libraryPLymphNode;
    public bool libraryPBoneMarrow;
    public bool libraryPFat;
    public bool libraryPCellWall;

    //enemyUnits
    public bool libraryEStreptococcusPneumoniae;
    public bool libraryEStaphyllococcusAureus;
    public bool libraryEInfluenza;
    public bool libraryEAnisakis;

    public PlayerData(Player player)
    {
        level = player.level;
        settingMusic = player.settingMusic;
        settingSound = player.settingSound;

        scoreExterminate = player.scoreExterminate;
        scoreDefend = player.scoreDefend;
        scoreSurvival = player.scoreSurvival;

        //playerUnits
        libraryPRedBloodCell = player.libraryPRedBloodCell;
        libraryPTrombocytes = player.libraryPTrombocytes;
        libraryPWhiteBloodCell = player.libraryPWhiteBloodCell;
        libraryPMacrophage = player.libraryPMacrophage;
        libraryPBCell = player.libraryPBCell;
        libraryPTCell = player.libraryPTCell;
        libraryPEosinophils = player.libraryPEosinophils;

        //playerBuildings
        libraryPNerveCenter = player.libraryPNerveCenter;
        libraryPIntestine = player.libraryPIntestine;
        libraryPArtery = player.libraryPArtery;
        libraryPLymphNode = player.libraryPLymphNode;
        libraryPBoneMarrow = player.libraryPBoneMarrow;
        libraryPFat = player.libraryPFat;
        libraryPCellWall = player.libraryPCellWall;

        //enemyUnits
        libraryEStreptococcusPneumoniae = player.libraryEStreptococcusPneumoniae;
        libraryEStaphyllococcusAureus = player.libraryEStaphyllococcusAureus;
        libraryEInfluenza = player.libraryEInfluenza;
        libraryEAnisakis = player.libraryEAnisakis;
    }
}
```

6. *SaveScript*

```
using UnityEngine;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

public static class SaveScript
{
    public static void Save(Player player)
    {
        BinaryFormatter formatter = new BinaryFormatter();
```

```
        string path = Application.persistentDataPath + "/player.ta";
        FileStream stream = new FileStream(path, FileMode.Create);

        PlayerData data = new PlayerData(player);

        formatter.Serialize(stream, data);
        stream.Close();

    }

    public static PlayerData LoadPlayer()
    {
        string path = Application.persistentDataPath + "/player.ta";
        if (File.Exists(path))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            FileStream stream = new FileStream(path,FileMode.Open);

            PlayerData data = formatter.Deserialize(stream) as PlayerData;
            stream.Close();

            return data;
        }
        else
        {
            Debug.LogError("Save File Not Found!" + path);
            return null;
        }
    }
}
```

7. *LibraryScript*

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

public class LibraryScript : MonoBehaviour
{
    public List<GameObject> libraryListPanel;
    public List<GameObject> libraryListGameObject;
    public List<bool> libraryListBool;
    // Start is called before the first frame update
    void Start()
    {
        Transform tmp = GameObject.Find("LibraryList").transform;
        foreach(Transform obj in tmp)
        {
            libraryListPanel.Add(obj.gameObject);
        }
        startOrganize();
        updateLibraryList();
    }


    void startOrganize()
    {
        for (int i = 0; i < libraryListGameObject.Count; i++)
        {
            RectTransform tmp = libraryListPanel[i].GetComponent<RectTransform>();
            tmp.anchorMin = new Vector2(0f,1f - (float)((i + 1) * (float)(1 / (float)libraryListGameObject.Count)));
            tmp.anchorMax = new Vector2(1f,1f - (float)(i * (float)(1 / (float)libraryListGameObject.Count)));
        }
    }

    public void updateLibraryList()
    {
        for (int i = 0; i < libraryListGameObject.Count; i++)
        {
            infoUnitOrBuilding(libraryListGameObject[i], i);
```

```
    }
}

void infoUnitOrBuilding(GameObject obj,int x)
{
    //OxygenColor
    Color oxygenCol;
    ColorUtility.TryParseHtmlString("#479EFF", out oxygenCol);
    Color cellCol;
    ColorUtility.TryParseHtmlString("#FF00D9", out cellCol);

    List<string> a = new List<string>();
    List<Color> b = new List<Color>();
    List<bool> c = new List<bool>();
    List<bool> d = new List<bool>();

    UnitStatus us = null;
    BuildingStatus bs = null;
    if (obj.GetComponent<UnitStatus>() != null)
        us = obj.GetComponent<UnitStatus>();
    else if (obj.GetComponent<BuildingStatus>() != null)
        bs = obj.GetComponent<BuildingStatus>();


    if (obj.GetComponent<UnitStatus>() != null)
    {
        if(obj.tag.Equals("PlayerSelectable"))
            addText(a, b, c, d, "[" + us.UnitName + "]", Color.yellow, true, true);
        else
            addText(a, b, c, d, "[" + us.UnitName + "]", new Color32(199, 105, 217,0), true, true);

        addText(a, b, c, d, "", Color.white, false, true);
        addText(a, b, c, d, "Description:", Color.white, true, true);
        addText(a, b, c, d, us.description, Color.white, false, true);
        addText(a, b, c, d, "", Color.white, false, true);
        addText(a, b, c, d, "Status:", Color.white, true, true);
        addText(a, b, c, d, "HP", Color.green, true, false);
        addText(a, b, c, d, ": " + us.MaxHealth, Color.white, false, true);
        addText(a, b, c, d, "Damage", Color.blue, true, false);
        addText(a, b, c, d, ": " + us.Damage + " - " + us.MaxDamage, Color.white, false, true);
        addText(a, b, c, d, "Damage Type", Color.red, true, false);
        addText(a, b, c, d, ": " + us.DamageType, Color.white, false, true);
        addText(a, b, c, d, "Attack Range", Color.cyan, true, false);
        if (us.Range <= 2)
            addText(a, b, c, d, ": Melee", Color.white, false, true);
        else
            addText(a, b, c, d, ": " + us.Range, Color.white, false, true);
        addText(a, b, c, d, "Attack Speed", Color.magenta, true, false);
        addText(a, b, c, d, ": " + us.attackSpeed, Color.white, false, true);
        addText(a, b, c, d, "Armour", Color.white, true, false);
        addText(a, b, c, d, ": " + us.Armour, Color.white, false, true);
        addText(a, b, c, d, "Armour Type", Color.gray, true, false);
        addText(a, b, c, d, ": " + us.ArmourType, Color.white, false, true);
        addText(a, b, c, d, "Movement Speed", new Color32(255,110,0,0), true, false);
        addText(a, b, c, d, ": " + obj.GetComponent<NavMeshAgent>().speed, Color.white, false, true);

        //setIcon
        libraryListPanel[x].transform.GetChild(0).GetComponent<Image>().sprite =
libraryListGameObject[x].GetComponent<UnitStatus>().iconSprite;
    }
    else if (obj.GetComponent<BuildingStatus>() != null)
    {
        addText(a, b, c, d, bs.BuildingName, Color.yellow, true, true);
        addText(a, b, c, d, "Require:", Color.white, false, true);
        if (bs.sugarReq > 0)
            addText(a, b, c, d, bs.sugarReq + " Sugar", Color.white, false, true);
        if (bs.oxygenReq > 0)
            addText(a, b, c, d, bs.oxygenReq + " Oxygen", oxygenCol, false, true);
        if (bs.cellsReq > 0)
            addText(a, b, c, d, bs.cellsReq + " Cells", cellCol, false, true);
        addText(a, b, c, d, "", Color.white, false, true);
        addText(a, b, c, d, "Description:", Color.white, false, true);
        addText(a, b, c, d, bs.description, Color.white, false, true);
```

```
            addText(a, b, c, d, "", Color.white, false, true);
            addText(a, b, c, d, "Build time: " + bs.buildTime + " seconds.", Color.gray, false, false);
            //setIcon
            libraryListPanel[x].transform.GetChild(0).GetComponent<Image>().sprite =
libraryListGameObject[x].GetComponent<BuildingStatus>().iconSprite;
        }

        setText(a, b, c, d, x);
    }

    public void addText(List<string> al, List<Color> bl, List<bool> cl, List<bool> dl, string a, Color b, bool c, bool
d)
    {
        al.Add(a);
        bl.Add(b);
        cl.Add(c);
        dl.Add(d);
    }


    private int textRow;
    public void setText(List<string> listString, List<Color> listColor, List<bool> bold, List<bool> enter,int x)
    {
        string tmp = "";
        textRow = 0;
        for (int i = 0; i < listString.Count; i++)
        {
            if (bold[i])
                tmp += "<b>";
            if (listString[i].Length > 30)
            {
                tmp += "<color=#" + ColorUtility.ToHtmlStringRGB(listColor[i]) + ">" + breakString(listString[i]) +
"</color>";
            }
            else
            {
                tmp += "<color=#" + ColorUtility.ToHtmlStringRGB(listColor[i]) + ">" + listString[i] + "</color>";
            }
            if (bold[i])
                tmp += "</b>";
            if (enter[i])
            {
                tmp += "\n";
                textRow++;
            }
        }
        //setText
        Text tex = libraryListPanel[x].transform.GetChild(1).GetChild(0).GetComponent<Text>();
        tex.text = tmp;
    }

    string breakString(string listString)
    {
        string tmp = "";
        int br = 0;
        for (int i = 0; i < listString.Length; i++)
        {
            tmp += listString.Substring(i, 1);
            if (br >= 30)
            {
                if (listString.Substring(i, 1).Equals(" "))
                {
                    br = 0;
                    tmp += "\n";
                    textRow++;
                }
            }
            br++;
        }
        return tmp;
    }

}
```

8. *SettingScript*

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

public class SettingScript : MonoBehaviour
{
    public int music;
    public int sound;
    private Player player;
    private Slider musicSlider;
    private Slider soundSlider;
    private TextMeshProUGUI musicText;
    private TextMeshProUGUI soundText;
    private Image applyButton;
    // Start is called before the first frame update
    void Start()
    {
        player = GameObject.Find("Player").GetComponent<Player>();
        musicSlider = GameObject.Find("musicSlider").GetComponent<Slider>();
        soundSlider = GameObject.Find("soundSlider").GetComponent<Slider>();
        musicText = GameObject.Find("txtMusic").GetComponent<TextMeshProUGUI>();
        soundText = GameObject.Find("txtSound").GetComponent<TextMeshProUGUI>();
        applyButton = GameObject.Find("settingApplyButton").GetComponent<Image>();
    }

    // Update is called once per frame
    void Update()
    {
        music = (int)musicSlider.value;
        sound = (int)soundSlider.value;
        musicText.text = "" + music;
        soundText.text = "" + sound;
        if (music != player.settingMusic || sound != player.settingSound)
            if(applyButton.color != Color.red)
                applyButton.color = Color.red;
    }

    public void loadSetting()
    {
        music = player.settingMusic;
        sound = player.settingSound;
        musicSlider.value = music;
        soundSlider.value = sound;
        musicText.text = "" + music;
        soundText.text = "" + sound;
        applyButton.color = Color.white;
    }

    public void applySetting()
    {
        player.settingMusic = music;
        player.settingSound = sound;
        applyButton.color = Color.white;
        player.SavePlayer();
    }
}
```

9. *SettingButtonScript*

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

public class SettingButtonScript : MonoBehaviour
{
    private SettingScript setting;
```

```
    public void onPressApply()
    {
        if (GameObject.Find("SettingMenu"))
        {
            setting = GameObject.Find("SettingMenu").GetComponent<SettingScript>();
            setting.applySetting();
        }
        else if(GameObject.Find("UIPauseMenu"))
        {
            setting = GameObject.Find("UIPauseMenu").GetComponent<SettingScript>();
            setting.applySetting();
        }

    }
    public void onPressLoad()
    {
        setting = GameObject.Find("SettingMenu").GetComponent<SettingScript>();
        setting.loadSetting();
    }
    public void onPressPause()
    {
        setting = GameObject.Find("UIPauseMenu").GetComponent<SettingScript>();
        setting.loadSetting();
        PlayerControls tmp = GameObject.Find("Player").GetComponent<PlayerControls>();
        tmp.Pause();
    }
}
```

10.*CameraMovement*

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

public class CameraMovement : MonoBehaviour
{
    public GameObject PlaneSize;
    public Transform MinimapCamBox;
    public float panSpeed = 0.3f; //Camera Movement Speed (0.3-1)
    public int camDistance = 10; //Distance of Camera (8-18)
    private float panDetect = 10f; //Detect Edge Screen Pixle (10f)


    private PlayerControls MapGlobal;


    private void Awake()
    {
        MapGlobal = gameObject.GetComponent<PlayerControls>();
        map = GameObject.Find("Map").GetComponent<GlobalAttribute>();
    }

    public Camera main;
    void Update()
    {
        if (MapGlobal.gamePause.Equals(false) && MapGlobal.cameraDisable.Equals(false))
        {
            MoveCamera();
        }
    }

    private GlobalAttribute map;
    //(GlobalAttribute)PlaneSize.GetComponent(typeof(GlobalAttribute))
    void MoveCamera()
    {
        float moveX = Camera.main.transform.position.x;
        float moveZ = Camera.main.transform.position.z;

        float xPos = Input.mousePosition.x;
        float yPos = Input.mousePosition.y;
```

```
        if (Input.GetKey(KeyCode.LeftArrow) ||
            xPos < panDetect)
        {
            if (Camera.main.transform.position.x > map.mapSizeWidth * 10 / 2 * -1)
                moveX -= panSpeed;
        }
        else if (Input.GetKey(KeyCode.RightArrow) ||
            xPos < Screen.width && xPos > Screen.width - panDetect)
        {
            if (Camera.main.transform.position.x < map.mapSizeWidth * 10 / 2)
                moveX += panSpeed;
        }

        if (Input.GetKey(KeyCode.UpArrow) ||
            yPos < Screen.height && yPos > Screen.height - panDetect)
        {
            if (Camera.main.transform.position.z < map.mapSizeHeight * 10 / 2)
                moveZ += (panSpeed * 2 / 3);
        }
        else if (Input.GetKey(KeyCode.DownArrow) ||
            yPos < panDetect)
        {
            if (Camera.main.transform.position.z > map.mapSizeHeight * 10 / 2 * -1)
                moveZ -= (panSpeed * 2 / 3);
        }

        //Camera Scroll In and Out to zoom
        int zoomSize = 10;
        if (Camera.main.orthographicSize >= camDistance && Camera.main.orthographicSize <= camDistance +
zoomSize)
        {
            Camera.main.orthographicSize -= (Input.GetAxis("Mouse ScrollWheel") * 10);
            if (Camera.main.orthographicSize < camDistance) Camera.main.orthographicSize = camDistance;
            else if (Camera.main.orthographicSize > camDistance + zoomSize) Camera.main.orthographicSize =
camDistance + zoomSize;
            MinimapCamBox.transform.localScale = new Vector3(Camera.main.orthographicSize / 0.8f,
Camera.main.orthographicSize / 0.8f,1);
        }
        panSpeed = Camera.main.orthographicSize / (camDistance + zoomSize) ;
        Vector3 newPos = new Vector3(moveX, 10, moveZ);
        Camera.main.transform.position = newPos;
    }
}
```

11.*ResourceManager*

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

public class ResourceManager : MonoBehaviour
{
    public int sugar;
    public int maxsugar;
    public int oxygen;
    public int maxoxygen;
    public int cells;
    public int maxcells;

}
```

12.*SystemNotifText*

```
using TMPro;
using UnityEngine;

public class SystemNotifText : MonoBehaviour
{
    private TextMeshProUGUI textMesh;
    private float disapearTimer;
```

```
    private Color textColor;
    private bool textMove;

    public static SystemNotifText Create(string txt,bool move, Color col, float dis)
    {
        Transform systemTextTransform = Instantiate(GameAssets.ins.pfSystemNotifText, new Vector3(-500f,0f,
0f), Quaternion.identity);
        SystemNotifText systemText = systemTextTransform.GetComponent<SystemNotifText>();
        systemText.Setup(txt, move, col, dis);
        return systemText;
    }
    private void Awake()
    {
        transform.SetParent(GameObject.Find("UIMain").transform);
        textMesh = transform.GetComponent<TextMeshProUGUI>();
        disapearTimer = 900f;
        textColor = textMesh.color;
    }

    public void Setup(string txt, bool move, Color col, float dis)
    {
        RectTransform rt = transform.GetComponent<RectTransform>();

        rt.sizeDelta = new Vector2(Screen.width - (Screen.width * 1.53f), Screen.height - (Screen.height * 1.96f));
        textMesh.text = txt;
        textMove = move;
        textColor = col;
        textMesh.color = col;
        disapearTimer = dis;

    }

    private float moveYSpeed = 0f;
    void Update()
    {
            RectTransform rt = transform.GetComponent<RectTransform>();

        if (textMove)
        {
            moveYSpeed += (Time.deltaTime * 23f);
        }
        transform.position = new Vector3(Screen.width / 2, Screen.width / 5.5f + moveYSpeed, 0f);
        disapearTimer -= Time.deltaTime;

        if (disapearTimer < 0)
        {
            float disapearSpeed = 1.5f;
            textColor.a -= disapearSpeed * Time.deltaTime;
            textMesh.color = textColor;
            if (textColor.a < 0)
            {
                Destroy(gameObject);
            }
        }

    }
}
```

13.*UIMovementButtonScript*

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

public class UIMovementButtonScript : MonoBehaviour
{
    private PlayerControls PlayerCon;
    private UITooltipScript UITooltipScri;

    private void Start()
    {
        PlayerCon = GameObject.Find("Player").GetComponent<PlayerControls>();
```

```
        UITooltipScri = GameObject.Find("UITooltip").GetComponent<UITooltipScript>();
    }
    //Attack,Move,Stop,Hold
    public void MovementCommand(int i)
    {
        switch (i)
        {
            //attack
            case 1:
                PlayerCon.playerCommand = "Attack";
                break;
            //attack
            case 2:
                PlayerCon.playerCommand = "Move";
                break;
            //attack
            case 3:
                PlayerCon.playerCommand = "None";
                PlayerCon.changeStatusSelectedUnits("Stop");
                break;
            //attack
            case 4:
                PlayerCon.changeStatusSelectedUnits("Hold");
                break;
        }
    }

    public void onHoverEnter(int i)
    {
        //Move, Attack, Stop, Hold
        List<string> a = new List<string>();
        List<Color> b = new List<Color>();
        List<bool> c = new List<bool>();
        List<bool> d = new List<bool>();

        switch (i)
        {
            case 1:
                UITooltipScri.addText(a, b, c, d, "Attack", Color.yellow, true, true);
                UITooltipScri.addText(a, b, c, d, "Target an Enemy or Ground to Command Selected Units to Attack
the Target or, Ground to AttackMove. When AttackMove the Selected Units Will Move to Target if the Unit
Detect a Nearby Enemy, then the Unit Will Attack.", Color.white, false, false);
                break;
            case 2:
                UITooltipScri.addText(a, b, c, d, "Move", Color.yellow, true, true);
                UITooltipScri.addText(a, b, c, d, "Move Selected Units to Target Ground.", Color.white, false, false);
                break;
            case 3:
                UITooltipScri.addText(a, b, c, d, "Stop", Color.yellow, true, true);
                UITooltipScri.addText(a, b, c, d, "Stop All Selected Units Command.", Color.white, false, false);
                break;
            case 4:
                UITooltipScri.addText(a, b, c, d, "Hold", Color.yellow, true, true);
                UITooltipScri.addText(a, b, c, d, "Stop All Selected Units Command and Hold the Ground, Will not
Attack Enemy.", Color.white, false, false);
                break;
        }

        UITooltipScri.setText(a, b, c, d);
    }
    public void onHoverExit()
    {
        UITooltipScri.HideTooltip();
    }
}
```

14.*GameAssets*

```
using UnityEngine;
using System.Reflection;

public class GameAssets : MonoBehaviour
```

```
{
    private static GameAssets _ins;

    public static GameAssets ins
    {
        get
        {
            if (_ins == null) _ins = (Instantiate(Resources.Load("GameAssets")) as
GameObject).GetComponent<GameAssets>();
            return _ins;
        }
    }

    public Transform pfFog;
    public Transform pfPlane;
    public Transform pfEffectMove;
    public Transform pfEffectDamagePopup;
    public Transform pfSystemNotifText;

    public Transform pfMissile;

}
```

15.*MultipleUnitsButton*

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

public class MultipleUnitsButton : MonoBehaviour
{
    public int ButtonNo;
    public GameObject UIUpdate;
    public void changeList()
    {
        //Debug.Log("Change " + ButtonNo);
        PlayerControls Script = GameObject.Find("Player").GetComponent<PlayerControls>();
        GameObject tmp = Script.SelectedUnits[ButtonNo];
        Script.SelectedUnits.RemoveAt(ButtonNo);
        Script.SelectedUnits.Insert(0, tmp);
        UIUpdate.GetComponent<UIScripts>().UIInfo(Script.SelectedUnits);
    }
}
```

16.*GlobalAttribute*

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

public class GlobalAttribute : MonoBehaviour
{
    public GameObject MinimapCam;

    //Game Status
    [Header("Game Status")]

    //Damage Calculator
    [Header("Damage Calculator")]
    public float armorDamageReduction = 0.05f; //For every 1 armor + 5% damage reduction
    //Ally
    [Header("Damage Ally")]
    public float cellToBacteria = 1.2f;
    public float cellToVirus = 0.7f;
    public float cellToFungi = 0.3f;
    public float cellToParasite = 0.5f;

    public float chemicalToBacteria = 1.5f;
    public float chemicalToVirus = 1.0f;
    public float chemicalToFungi = 0.5f;
    public float chemicalToParasite = 0.3f;
```

```
public float viralToBacteria = 0.8f;
public float viralToVirus = 0.8f;
public float viralToFungi = 0.2f;
public float viralToParasite = 2.0f;

//Enemy
[Header("Damage Enemy")]
public float bacteriaToCell = 1.0f;
public float bacteriaToBuilding = 0.7f;

public float virusToCell = 1.2f;
public float virusToBuilding = 0.8f;

public float fungiToCell = 0.6f;
public float fungiToBuilding = 0.0f;

public float parasiteToCell = 1.2f;
public float parasiteToBuilding = 2.0f;


public int damageCalculator(int damage, string myType, string targetType, int targetArmor)
{
    int outDamage = 0;
    float damagePercent = 0;
    outDamage = damage;

    //Damage Reduction
    //Ally
    if (myType.Equals("Cell") && targetType.Equals("Bacteria"))
        damagePercent = cellToBacteria;
    else if (myType.Equals("Cell") && targetType.Equals("Virus"))
        damagePercent = cellToVirus;
    else if (myType.Equals("Cell") && targetType.Equals("Fungi"))
        damagePercent = cellToFungi;
    else if (myType.Equals("Cell") && targetType.Equals("Parasite"))
        damagePercent = cellToParasite;
    else if (myType.Equals("Chemical") && targetType.Equals("Bacteria"))
        damagePercent = chemicalToBacteria;
    else if (myType.Equals("Chemical") && targetType.Equals("Virus"))
        damagePercent = chemicalToVirus;
    else if (myType.Equals("Chemical") && targetType.Equals("Fungi"))
        damagePercent = chemicalToFungi;
    else if (myType.Equals("Chemical") && targetType.Equals("Parasite"))
        damagePercent = chemicalToParasite;
    else if (myType.Equals("Viral") && targetType.Equals("Bacteria"))
        damagePercent = viralToBacteria;
    else if (myType.Equals("Viral") && targetType.Equals("Virus"))
        damagePercent = viralToVirus;
    else if (myType.Equals("Viral") && targetType.Equals("Fungi"))
        damagePercent = viralToFungi;
    else if (myType.Equals("Viral") && targetType.Equals("Parasite"))
        damagePercent = viralToParasite;
    //Enemy
    else if (myType.Equals("Bacteria") && targetType.Equals("Cell"))
        damagePercent = bacteriaToCell;
    else if (myType.Equals("Bacteria") && targetType.Equals("Building"))
        damagePercent = bacteriaToBuilding;
    else if (myType.Equals("Virus") && targetType.Equals("Cell"))
        damagePercent = virusToCell;
    else if (myType.Equals("Virus") && targetType.Equals("Building"))
        damagePercent = virusToBuilding;
    else if (myType.Equals("Fungi") && targetType.Equals("Cell"))
        damagePercent = fungiToCell;
    else if (myType.Equals("Fungi") && targetType.Equals("Building"))
        damagePercent = fungiToBuilding;
    else if (myType.Equals("Parasite") && targetType.Equals("Cell"))
        damagePercent = parasiteToCell;
    else if (myType.Equals("Parasite") && targetType.Equals("Building"))
        damagePercent = parasiteToBuilding;

    //Count Armor
    damagePercent -= (targetArmor * armorDamageReduction);
    if (damagePercent < 0)
```

```
            damagePercent = 0;
        //Calculate Damage Dealt
        outDamage = (int)(float)(outDamage * damagePercent);
        Debug.Log("DP:"+damagePercent + "% | Damage:" + outDamage +" Dealt!");
        return outDamage;
    }

    //Create Map
    [HideInInspector] public int mapSizeWidth;
    [HideInInspector] public int mapSizeHeight;
    public GameObject[,] ReCreateMap(GameObject[,] terrainCrossID)
    {
        terrainCrossID = new GameObject[mapSizeWidth, mapSizeHeight];
        if (GameObject.Find("FogGroup")|| GameObject.Find("PanelGroup"))
        {
            DestroyImmediate(GameObject.Find("FogGroup"));
            DestroyImmediate(GameObject.Find("PanelGroup"));
            DestroyImmediate(GameObject.Find("GameAssets(Clone)"));
        }
        else
            Debug.Log("not find");
        float largestSize = 0;
        if (mapSizeWidth > mapSizeHeight)
        {
            largestSize = mapSizeWidth;
        }
        else
        {
            largestSize = mapSizeHeight;
        }
        //Panel And Fog
        GameObject FG = new GameObject("FogGroup");
        GameObject PG = new GameObject("PanelGroup");
        MinimapCam.GetComponent<Camera>().orthographicSize = largestSize * 5;
        for (int x = 0; x < mapSizeWidth; x++)
        {
            for (int y = 0; y < mapSizeHeight; y++)
            {
                Transform f = Instantiate(GameAssets.ins.pfFog, new Vector3(
                    x * 10f - mapSizeWidth  * 5 +5, 1.5f,
                    y * 10f - mapSizeHeight * 5 + 5), Quaternion.identity);
                Transform p = Instantiate(GameAssets.ins.pfPlane, new Vector3(
                    x * 10f - mapSizeWidth * 5 + 5, 0f,
                    y * 10f - mapSizeHeight * 5 + 5), Quaternion.identity);
                terrainCrossID[x, y] = p.gameObject;
                p.GetComponent<TerrainScript>().terrainID(x,y);
                f.transform.eulerAngles = new Vector3(90f, 0, 0);
                //p.transform.eulerAngles = new Vector3(0, 180f, 0);

                f.transform.parent = FG.transform;
                p.transform.parent = PG.transform;
            }
        }
        FG.transform.parent = GameObject.Find("Map").transform;
        PG.transform.parent = GameObject.Find("Map").transform;
        return terrainCrossID;
    }

    public GameObject[,] ReMarkTerrainID()
    {
        GameObject[,] tmp = new GameObject[mapSizeWidth, mapSizeHeight];
        foreach (Transform child in GameObject.Find("PanelGroup").transform)
        {
            if (child.GetComponent<TerrainScript>() != null)
            {
                tmp[child.GetComponent<TerrainScript>().tIDX, child.GetComponent<TerrainScript>().tIDY] =
child.gameObject;
            }
        }

        return tmp;
    }
    //HideFog
```

```
    public void HideFog()
    {
        foreach (Transform child in transform.Find("FogGroup"))
        {
            child.GetComponent<SpriteRenderer>().enabled = !child.GetComponent<SpriteRenderer>().enabled;
        }
    }
}
```

17.*DamagePopup*

```
using UnityEngine;
using TMPro;

public class DamagePopup : MonoBehaviour
{
    private TextMeshPro textMesh;
    private RectTransform rectTransform;
    private float disapearTimer;
    private Color textColor;

    public static DamagePopup Create(Vector3 position, string txt, int damageAmount, byte cr, byte cg, byte cb,
byte ct)
    {
        Transform damagePopupTransoform = Instantiate(GameAssets.ins.pfEffectDamagePopup, new
Vector3(position.x + Random.Range(-0.3f,0.3f), position.y, position.z + 2.1f), Quaternion.identity);
        DamagePopup damagePopup = damagePopupTransoform.GetComponent<DamagePopup>();
        damagePopup.Setup(txt,damageAmount,cr,cg,cb,ct);
        return damagePopup;
    }

    private void Awake()
    {
        textMesh = transform.GetComponent<TextMeshPro>();
        rectTransform = transform.GetComponent<RectTransform>();
        moveXSpeed = Random.Range(-0.6f, 0.6f);
    }

    public void Setup(string txt,int damageAmount,byte cr, byte cg, byte cb, byte ct)
    {
        textMesh.SetText(txt+damageAmount.ToString());
        textMesh.color = new Color32(cr,cg,cb,ct);
        rectTransform.Rotate(new Vector3(90, 0, 0));
        textColor = textMesh.color;
        disapearTimer = 1f;
    }

    private float moveXSpeed;
    private void Update()
    {

        float moveZSpeed = 1.2f;
        transform.position += new Vector3(moveXSpeed, 0,moveZSpeed) * Time.deltaTime;
        disapearTimer -= Time.deltaTime;
        if (disapearTimer < 0)
        {
            float disapearSpeed = 3f;
            textColor.a -= disapearSpeed * Time.deltaTime;
            textMesh.color = textColor;
            if (textColor.a < 0)
            {
                Destroy(gameObject);
            }
        }
    }
}
```

18.*MissileScript*

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

```
using UnityEngine.AI;

public class MissileScript : MonoBehaviour
{
    public GameObject target;
    public int damage;
    public Vector3 point;
    public Vector3 originalPoint;

    public float time;

    public GameObject sprite;


    public static MissileScript Create(GameObject Tar, int Dam, Vector3 Position, float Time, Sprite Sprite)
    {
        Transform transform = Instantiate(GameAssets.ins.pfMissile, new Vector3(Position.x, Position.y,
Position.z), Quaternion.identity);
        MissileScript script = transform.GetComponent<MissileScript>();
        script.Setup(Tar, Dam, Position, Time, Sprite);
        return script;
    }

    private void Awake()
    {
        originalPoint = gameObject.transform.position;
        //Up face the target
    }

    public void Setup(GameObject Tar, int Dam, Vector3 Position, float Time, Sprite Sprite)
    {
        target = Tar;
        damage = Dam;
        point = TransformPointRangeAngle(new Vector3(Tar.transform.position.x, 0,
Tar.transform.position.z),1.6f,0);
        time = Time;
        createSprite(Sprite);
    }

    void createSprite(Sprite Sprite)
    {
        sprite = new GameObject("Sprite");
        sprite.AddComponent<SpriteRenderer>();
        sprite.GetComponent<SpriteRenderer>().sprite = Sprite;
        sprite.transform.localPosition = new Vector3(gameObject.transform.position.x,2,
gameObject.transform.position.z);
        sprite.transform.parent = gameObject.transform;
        sprite.transform.localRotation = Quaternion.Euler(90, CalculateAngle(originalPoint, point),
gameObject.transform.localRotation.z);
    }

    // Update is called once per frame
    void Update()
    {
        moveToPoint();
    }

    void moveToPoint()
    {
        if ((gameObject.transform.position.x > point.x - 0.1f && gameObject.transform.position.x < point.x + 0.1f) ||
            (gameObject.transform.position.z > point.z - 0.1f && gameObject.transform.position.z < point.z + 0.1f))
        {
            //Activate Damage
            if (target != null)
            {
                if (target.GetComponent<UnitStatus>() != null)
                {
                    target.GetComponent<UnitStatus>().Health -= damage;
                }
                else if (target.GetComponent<BuildingStatus>() != null)
                {
                    target.GetComponent<BuildingStatus>().Health -= damage;
                }
```

```
            if (target.tag == "EnemySelectable")
                DamagePopup.Create(target.transform.position, "-", damage, 255, 0, 0, 255);
            else
                DamagePopup.Create(target.transform.position, "-", damage, 0, 100, 0, 255);

            //If Builder Gets Damaged Run Away
            if (target.GetComponent<UnitStatus>() != null)
            {
                if (target.GetComponent<UnitStatus>().isBuilder && target.GetComponent<UnitStatus>().Target ==
null)
                {
                    Vector3 tmp = TransformPointRangeAngle(target.transform.position, 4.5f,
CalculateAngle(gameObject.transform.position, target.transform.position));
                    target.GetComponent<UnitStatus>().State = "Move";
                    target.GetComponent<UnitStatus>().Target = null;
                    target.GetComponent<UnitStatus>().checkStates = false;
                    target.GetComponent<UnitStatus>().flipReset = true;
                    target.GetComponent<UnitStatus>().movePoint = tmp;
                    target.GetComponent<NavMeshAgent>().destination = tmp;
                }
            }

            DestroyMe();
        }
        else
        {
            DestroyMe();
        }
    }
    else
        gameObject.transform.position = new Vector3(gameObject.transform.position.x + ((point.x -
originalPoint.x) / time / 60)
                                    , gameObject.transform.position.y
                                    , gameObject.transform.position.z + ((point.z - originalPoint.z) / time / 60));

  }

  Vector3 TransformPointRangeAngle(Vector3 startPoint, float distance, float angle)
  {
    Transform poi = new GameObject().transform;
    poi.position = startPoint;
    poi.localRotation = Quaternion.Euler(0, angle, 0);
    poi.Translate(0, 0, distance);
    Vector3 endPoint = poi.position;
    Destroy(poi.gameObject);
    return endPoint;
  }
  float CalculateAngle(Vector3 from, Vector3 to)
  {
    return (Mathf.Atan2(from.z - to.z, to.x - from.x) * Mathf.Rad2Deg) + 90f;
  }

  public void DestroyMe()
  {
    Destroy(gameObject);
  }
}
```

19.*UIScripts*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine.AI;
using UnityEngine;
using TMPro;

public class UIScripts : MonoBehaviour
{
  public List<Camera> ProfilePic;
  public List<RenderTexture> ProfilePicUnits;
  public List<GameObject> SPUStats; //Single Panel Unit
```

```
    public List<TextMeshProUGUI> MPUText; //Multi Panel Units
    public List<TextMeshProUGUI> MPUName;
    public List<RawImage> MPUIcon;
    public List<Button> MPUButton;
    public List<GameObject> MPResource; //0-2 txt|3-5 panel


    //Main Select
    private float[] PPTime = new float[5];//Profile Pic Timer
    public void UIInfo(List<GameObject> SelectedUnits)
    {
        //if Select Unit
        if (SelectedUnits.Count > 0 && SelectedUnits[0].gameObject.GetComponent<UnitStatus>() != null)
        {
            getallchild(GameObject.Find("MovementPan").transform, true);
            GameObject.Find("RawImageProfile").GetComponent<RawImage>().enabled = true;
            if (SelectedUnits.Count == 1)
            {
                hideBotUI(true, false, false);
                ProfilePic[0].gameObject.transform.position = new Vector3(
                    SelectedUnits[0].gameObject.transform.position.x, 5,
SelectedUnits[0].gameObject.transform.position.z + 2f);
                ProfilePic[0].enabled = true;
                //Name
                SPUStats[0].GetComponent<TextMeshProUGUI>().text = "" +
SelectedUnits[0].gameObject.GetComponent<UnitStatus>().UnitName;
                //Damage
                SPUStats[1].GetComponent<TextMeshProUGUI>().text =
SelectedUnits[0].gameObject.GetComponent<UnitStatus>().Damage + "-" +
SelectedUnits[0].gameObject.GetComponent<UnitStatus>().MaxDamage;
                //Armor
                SPUStats[2].GetComponent<TextMeshProUGUI>().text = "" +
SelectedUnits[0].gameObject.GetComponent<UnitStatus>().Armour;
                //Speed
                SPUStats[3].GetComponent<TextMeshProUGUI>().text = "" +
SelectedUnits[0].gameObject.GetComponent<NavMeshAgent>().speed;

            }
            else
            {
                hideBotUI(false, true, false);
                List<string> nameList = new List<string>();
                List<int> countList = new List<int>();


                //Add Selected Unit to List
                for (int i = 0; i < SelectedUnits.Count; i++)
                {
                    if (!nameList.Contains(SelectedUnits[i].gameObject.GetComponent<UnitStatus>().UnitName)&&
nameList.Count<5)
                    {
                        nameList.Add(SelectedUnits[i].gameObject.GetComponent<UnitStatus>().UnitName);
                        countList.Add(0);
                    }
                }
                //Count every selected units with the same name
                for (int i = 0; i < nameList.Count; i++)
                {
                    foreach (GameObject obj in SelectedUnits)
                    {
                        if (obj.gameObject.GetComponent<UnitStatus>().UnitName.Equals(nameList[i]))
                        {
                            if (i < 5)
                            {
                                ProfilePic[i].gameObject.transform.position = new Vector3(
                                    SelectedUnits[i].gameObject.transform.position.x, 5,
                                    SelectedUnits[i].gameObject.transform.position.z + 2f);
                                ProfilePic[i].enabled = true;

                            }
                            countList[i]++;
                        }
                    }
```

```
            }

            //Name
            SPUStats[0].GetComponent<TextMeshProUGUI>().text =
SelectedUnits[0].gameObject.GetComponent<UnitStatus>().UnitName + " (" + countList[0] + ")";
            for (int i = 0; i < MPUText.Count; i++)
            {
                MPUText[i].enabled = false;
                MPUName[i].enabled = false;
                MPUIcon[i].enabled = false;
                MPUButton[i].enabled = false;
            }

            for (int i = 1; i < nameList.Count; i++)
            {
                MPUText[i - 1].enabled = true;
                MPUName[i - 1].enabled = true;
                MPUIcon[i - 1].enabled = true;
                MPUButton[i - 1].enabled = true;
                //Number of Units
                MPUText[i - 1].text = "x" + countList[i];
                //Names
                MPUName[i - 1].text = nameList[i];
            }
        }
    }
    //if Select Building
    else if (SelectedUnits.Count > 0 && SelectedUnits[0].gameObject.GetComponent<BuildingStatus>() !=
null)
    {
        getallchild(GameObject.Find("MovementPan").transform, false);
    }
    else if (SelectedUnits.Count <= 0)
    {
        hideBotUI(false, false, true);
        SPUStats[0].GetComponent<TextMeshProUGUI>().text = "";
        GameObject tmpUI = GameObject.Find("panelHP");
        tmpUI.GetComponent<RectTransform>().localScale = new Vector3(0,1,1);
        tmpUI = GameObject.Find("txtProfile");
        tmpUI.GetComponent<TextMeshProUGUI>().text = "0 / 0";
        tmpUI = GameObject.Find("RawImageProfile");
        tmpUI.GetComponent<RawImage>().enabled = false;

    }


}
public void hideBotUI(bool sin, bool mul, bool emp)
{
    getallchild(GameObject.Find("BotPanelSingle").transform, sin);
    getallchild(GameObject.Find("BotPanelMultiple").transform, mul);
    getallchild(GameObject.Find("EmptyPanel").transform, emp);
}

void getallchild(Transform p, bool t)
{
    if (p.GetComponent<Image>() != null)
        p.GetComponent<Image>().enabled = t;
    if (p.GetComponent<TextMeshProUGUI>() != null)
        p.GetComponent<TextMeshProUGUI>().enabled = t;
    if (p.GetComponent<RawImage>() != null)
        p.GetComponent<RawImage>().enabled = t;
    if (p.GetComponent<Button>() != null)
        p.GetComponent<Button>().enabled = t;
    if (p.transform.childCount > 0)
    {
        foreach (Transform child in p)
        {
            if (child.GetComponent<Image>() != null)
                child.GetComponent<Image>().enabled = t;
            if (child.GetComponent<TextMeshProUGUI>() != null)
                child.GetComponent<TextMeshProUGUI>().enabled = t;
            if (child.GetComponent<RawImage>() != null)
```

```
                child.GetComponent<RawImage>().enabled = t;
            if (child.GetComponent<Button>() != null)
                child.GetComponent<Button>().enabled = t;
            getallchild(child, t);
        }
    }
}
public void UIHealth(List<GameObject> SelectedUnits)
{
    if (SelectedUnits.Count == 1)
    {
        if (SelectedUnits[0].gameObject.GetComponent<UnitStatus>() != null)
        {
            //Health
            GameObject tmpUI = GameObject.Find("panelHP");
            tmpUI.GetComponent<RectTransform>().localScale = new Vector3(
                (float)SelectedUnits[0].gameObject.GetComponent<UnitStatus>().Health /
                (float)SelectedUnits[0].gameObject.GetComponent<UnitStatus>().MaxHealth, 1, 1);
            tmpUI = GameObject.Find("txtProfile");
            tmpUI.GetComponent<TextMeshProUGUI>().text =
SelectedUnits[0].gameObject.GetComponent<UnitStatus>().Health
                + " / " + SelectedUnits[0].gameObject.GetComponent<UnitStatus>().MaxHealth;

        }
    }
    else if (SelectedUnits.Count > 1)
    {

        if (SelectedUnits[0].gameObject.GetComponent<UnitStatus>() != null)
        {
            //Health
            GameObject tmpUI = GameObject.Find("panelHP");
            tmpUI.GetComponent<RectTransform>().localScale = new Vector3(
                (float)SelectedUnits[0].gameObject.GetComponent<UnitStatus>().Health /
                (float)SelectedUnits[0].gameObject.GetComponent<UnitStatus>().MaxHealth, 1, 1);
            tmpUI = GameObject.Find("txtProfile");
            tmpUI.GetComponent<TextMeshProUGUI>().text =
SelectedUnits[0].gameObject.GetComponent<UnitStatus>().Health
                + " / " + SelectedUnits[0].gameObject.GetComponent<UnitStatus>().MaxHealth;

        }
    }

    for (int i = 0; i < PPTime.Length; i++)
    {
        if (ProfilePic[i].enabled)
        {
            if (PPTime[i] > 0f)
            {
                ProfilePic[i].enabled = false;
                PPTime[i] = 0;
            }
            else
            {
                PPTime[i] += Time.deltaTime;
            }
        }
    }
}

private ResourceManager ResourceM;
private List<int> resource;
public void UIResource()
{

    resource = new List<int>();
    resource.Add(ResourceM.sugar);
    resource.Add(ResourceM.oxygen);
    resource.Add(ResourceM.cells);
    resource.Add(ResourceM.maxsugar);
    resource.Add(ResourceM.maxoxygen);
    resource.Add(ResourceM.maxcells);
    for (int i=0; i<3; i++)
```

```
        {
            MPResource[i].GetComponent<TextMeshProUGUI>().text = resource[i].ToString() + "/" +
resource[i+3].ToString();
            float x = 0;
            if (resource[i + 3] > 0)
                x = (float)resource[i] / (float)resource[i + 3];
            MPResource[i+3].GetComponent<RectTransform>().localScale = new Vector3(
                    x, 1, 1);
            if (i < 2)
            {
                if (resource[i] < resource[i + 3] / 4)
                    MPResource[i + 3].GetComponent<Image>().color = Color.red;
                else if (resource[i] > resource[i + 3] / 2)
                    MPResource[i + 3].GetComponent<Image>().color = Color.green;
                else
                    MPResource[i + 3].GetComponent<Image>().color = Color.yellow;
            }
            else
            {
                if (resource[i] < resource[i + 3] / 4)
                    MPResource[i + 3].GetComponent<Image>().color = Color.green;
                else if (resource[i] > resource[i + 3] / 2)
                    MPResource[i + 3].GetComponent<Image>().color = Color.red;
                else
                    MPResource[i + 3].GetComponent<Image>().color = Color.yellow;
            }
        }
    }
    public void Start()
    {
        //ResourceManager
        ResourceM = GameObject.Find("Player").GetComponent<ResourceManager>();

        Transform tmp = GameObject.Find("BotPanelMultiple").transform;
        foreach (Transform obj in tmp)
        {
            MPUName.Add(obj.GetChild(0).GetComponent<TextMeshProUGUI>());
            MPUText.Add(obj.GetChild(1).GetComponent<TextMeshProUGUI>());
            MPUIcon.Add(obj.GetChild(2).GetChild(0).GetComponent<RawImage>());
            MPUButton.Add(obj.GetChild(2).GetChild(1).GetComponent<Button>());
        }
        for(int i = 1; i < 5; i++)
        {
            MPUIcon[i - 1].texture = ProfilePicUnits[i];
            MPUButton[i - 1].gameObject.GetComponent<MultipleUnitsButton>().ButtonNo = i;
        }
    }
    public void Update()
    {
        UIResource();
    }
}
```

20.*UIBuildTrainButton*

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

public class UIBuildTrainButton : MonoBehaviour
{
    public GameObject statusGameObject;

    private UITooltipScript tooltip;
    private ResourceManager ResourceM;
    private PlayerControls playerMoveC;

    private void Awake()
    {
        tooltip = GameObject.Find("UITooltip").GetComponent<UITooltipScript>();
        ResourceM = GameObject.Find("Player").GetComponent<ResourceManager>();
```

```
    playerMoveC = GameObject.Find("Player").GetComponent<PlayerControls>();
    readGameObject();
}
public void readGameObject()
{
    Transform parent = transform.parent;
    if (statusGameObject != null)
    {

        UnitStatus us = null;
        BuildingStatus bs = null;
        if (statusGameObject.GetComponent<UnitStatus>() != null)
            us = statusGameObject.GetComponent<UnitStatus>();
        else if (statusGameObject.GetComponent<BuildingStatus>() != null)
            bs = statusGameObject.GetComponent<BuildingStatus>();

        if (statusGameObject.GetComponent<UnitStatus>() != null)
        {
            if (us.sugarReq == 0)
                parent.GetChild(1).GetComponent<Text>().text = "";
            else
                parent.GetChild(1).GetComponent<Text>().text = "" + us.sugarReq;
            if (us.oxygenReq == 0)
                parent.GetChild(2).GetComponent<Text>().text = "";
            else
                parent.GetChild(2).GetComponent<Text>().text = "" + us.oxygenReq;
            if (us.cellsReq == 0)
                parent.GetChild(3).GetComponent<Text>().text = "";
            else
                parent.GetChild(3).GetComponent<Text>().text = "" + us.cellsReq;

            gameObject.GetComponent<Image>().sprite = us.iconSprite;
        }
        else if (statusGameObject.GetComponent<BuildingStatus>() != null)
        {

            if (bs.sugarReq == 0)
                parent.GetChild(1).GetComponent<Text>().text = "";
            else
                parent.GetChild(1).GetComponent<Text>().text = "" + bs.sugarReq;
            if (bs.oxygenReq == 0)
                parent.GetChild(2).GetComponent<Text>().text = "";
            else
                parent.GetChild(2).GetComponent<Text>().text = "" + bs.oxygenReq;
            if (bs.cellsReq == 0)
                parent.GetChild(3).GetComponent<Text>().text = "";
            else
                parent.GetChild(3).GetComponent<Text>().text = "" + bs.cellsReq;

            gameObject.GetComponent<Image>().sprite = bs.iconSprite;
            gameObject.GetComponent<Image>().enabled = true;
        }
    }
    else
    {
        parent.GetChild(1).GetComponent<Text>().text = "";
        parent.GetChild(2).GetComponent<Text>().text = "";
        parent.GetChild(3).GetComponent<Text>().text = "";
        gameObject.GetComponent<Image>().sprite = null;
        gameObject.GetComponent<Image>().enabled = false;
    }
}

void tooltipUnitOrBuilding()
{
    //OxygenColor
    Color oxygenCol;
    ColorUtility.TryParseHtmlString("#479EFF", out oxygenCol);
    Color cellCol;
    ColorUtility.TryParseHtmlString("#FF00D9", out cellCol);

    List<string> a = new List<string>();
    List<Color> b = new List<Color>();
```

```
        List<bool> c = new List<bool>();
        List<bool> d = new List<bool>();

        UnitStatus us = null;
        BuildingStatus bs = null;
        if (statusGameObject.GetComponent<UnitStatus>() != null)
            us = statusGameObject.GetComponent<UnitStatus>();
        else if (statusGameObject.GetComponent<BuildingStatus>() != null)
            bs = statusGameObject.GetComponent<BuildingStatus>();


        if (statusGameObject.GetComponent<UnitStatus>() != null)
        {
            tooltip.addText(a, b, c, d, us.UnitName, Color.yellow, true, true);
            tooltip.addText(a, b, c, d, "Require:", Color.white, false, true);
            if(us.sugarReq>0)
                tooltip.addText(a, b, c, d, us.sugarReq + " Sugar", Color.white, false, true);
            if (us.oxygenReq > 0)
                tooltip.addText(a, b, c, d, us.oxygenReq + " Oxygen", oxygenCol, false, true);
            if (us.cellsReq > 0)
                tooltip.addText(a, b, c, d, us.cellsReq + " Cells", cellCol, false, true);
            tooltip.addText(a, b, c, d, "", Color.white, false, true);
            tooltip.addText(a, b, c, d, "Description:", Color.white, false, true);
            tooltip.addText(a, b, c, d, us.description, Color.white, false, true);
            tooltip.addText(a, b, c, d, "", Color.white, false, true);
            tooltip.addText(a, b, c, d, "Train time: " + us.trainTime + " seconds.", Color.gray, false, false);
        }
        else if (statusGameObject.GetComponent<BuildingStatus>() != null)
        {
            tooltip.addText(a, b, c, d, bs.BuildingName, Color.yellow, true, true);
            tooltip.addText(a, b, c, d, "Require:", Color.white, false, true);
            if (bs.sugarReq > 0)
                tooltip.addText(a, b, c, d, bs.sugarReq + " Sugar", Color.white, false, true);
            if (bs.oxygenReq > 0)
                tooltip.addText(a, b, c, d, bs.oxygenReq + " Oxygen", oxygenCol, false, true);
            if (bs.cellsReq > 0)
                tooltip.addText(a, b, c, d, bs.cellsReq + " Cells", cellCol, false, true);
            tooltip.addText(a, b, c, d, "", Color.white, false, true);
            tooltip.addText(a, b, c, d, "Description:", Color.white, false, true);
            tooltip.addText(a, b, c, d, bs.description, Color.white, false, true);
            tooltip.addText(a, b, c, d, "", Color.white, false, true);
            tooltip.addText(a, b, c, d, "Build time: " + bs.buildTime + " seconds.", Color.gray, false, false);
        }


        tooltip.setText(a, b, c, d);
    }

    public void onPress(int i)
    {
        playerMoveC.buildButton(i);
    }
    public void onHoverEnter()
    {
        if (statusGameObject != null)
            if (statusGameObject.GetComponent<UnitStatus>() != null ||
statusGameObject.GetComponent<BuildingStatus>() != null)
                tooltipUnitOrBuilding();
    }
    public void onHoverExit()
    {
        tooltip.HideTooltip();
    }
}
```

21.*UITooltipScript*

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

```
public class UITooltipScript : MonoBehaviour
{
    private Text tooltipText;
    private RectTransform bgRectTransform;
    private RectTransform tooltipTransform;

    private void Awake()
    {
        bgRectTransform = transform.Find("BackgroundTooltip").GetComponent<RectTransform>();
        tooltipText = transform.Find("txtTooltip").GetComponent<Text>();
        tooltipTransform = transform.Find("txtTooltip").GetComponent<RectTransform>();

    }
    private void Start()
    {
        gameObject.SetActive(false);
    }

    public void addText(List<string> al, List<Color> bl, List<bool> cl, List<bool> dl, string a, Color b, bool c, bool
d)
    {
        al.Add(a);
        bl.Add(b);
        cl.Add(c);
        dl.Add(d);
    }


    private int textRow;
    public void setText(List<string> listString, List<Color> listColor, List<bool> bold, List<bool> enter)
    {
        string tmp = "";
        textRow = 0;
        for (int i = 0; i < listString.Count; i++)
        {
            if (bold[i])
                tmp += "<b>";
            if (listString[i].Length > 30)
            {
                tmp += "<color=#" + ColorUtility.ToHtmlStringRGB(listColor[i]) + ">" + breakString(listString[i]) +
"</color>";
            }
            else
            {
                tmp += "<color=#" + ColorUtility.ToHtmlStringRGB(listColor[i]) + ">" + listString[i] + "</color>";
            }
            if (bold[i])
                tmp += "</b>";
            if (enter[i])
            {
                tmp += "\n";
                textRow++;
            }
        }
        ShowTooltip(tmp);
        ShowTooltip(tmp);
    }

    string breakString(string listString)
    {
        string tmp = "";
        int br = 0;
        for(int i = 0; i<listString.Length; i++)
        {
            tmp += listString.Substring(i, 1);
            if (br >= 30)
            {
                if (listString.Substring(i, 1).Equals(" "))
                {
                    br = 0;
                    tmp += "\n";
                    textRow++;
                }
```

```
            }
            br++;
        }
        return tmp;
    }

    private Vector2 pos;
    private void Update()
    {
        if (transform.position.x + bgRectTransform.sizeDelta.x >= Screen.width)
        {
            if (transform.position.y + bgRectTransform.sizeDelta.y >= Screen.height)
            {
                transform.localScale = new Vector3(-1, -1, 1);
                tooltipTransform.localScale = new Vector3(-1, -1, 1);
            }
            else
            {
                transform.localScale = new Vector3(-1, 1, 1);
                tooltipTransform.localScale = new Vector3(-1, 1, 1);
            }
        }
        else
        {
            if (transform.position.y + bgRectTransform.sizeDelta.y >= Screen.height)
            {
                transform.localScale = new Vector3(1, -1, 1);
                tooltipTransform.localScale = new Vector3(1, -1, 1);
            }
            else
            {
                transform.localScale = new Vector3(1, 1, 1);
                tooltipTransform.localScale = new Vector3(1, 1, 1);
            }
        }

        transform.position = new Vector2(Input.mousePosition.x, Input.mousePosition.y);
    }

    private void ShowTooltip(string s)
    {
        gameObject.SetActive(true);

        tooltipText.text = s;
        float addSize = 4f;
        Vector2 bgSize = new Vector2(tooltipText.preferredWidth + addSize * 2, tooltipText.preferredHeight +
addSize * 2);
        bgRectTransform.sizeDelta = bgSize;

        tooltipTransform.sizeDelta = new Vector2(bgSize.x - 8, bgSize.y - 8);
        tooltipTransform.localPosition = new Vector2(bgSize.x / 2, bgSize.y / 2);
    }
    public void HideTooltip()
    {
        gameObject.SetActive(false);
    }
}
```

22.*MapFogScript*

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



public class MapFogScript : MonoBehaviour
{
    //public List<GameObject> insideL= new List<GameObject>();

    private List<GameObject> List = new List<GameObject>();
    private void OnTriggerEnter(Collider other)
```

```
    {
        if (other.gameObject.tag == "PlayerSelectable" && !other.isTrigger)
        {
            List.Add(other.gameObject);
            tmp = true;
            changeAlpha();
        }
    }
    private void OnTriggerExit(Collider other)
    {
        if (List.Contains(other.gameObject))
        {
            List.Remove(other.gameObject);
            tmp = true;
            changeAlpha();
        }
    }

    private void LateUpdate()
    {
        checkListMissing();
        changeAlpha();
    }

    private bool tmp = false;
    void changeAlpha()
    {
        if (tmp)
        {
            if (List.Count > 0)
                gameObject.GetComponent<SpriteRenderer>().color = new Color32(255, 255, 255, 0);
            else
                gameObject.GetComponent<SpriteRenderer>().color = new Color32(255, 255, 255, 255);
            tmp = false;
        }
    }

    void checkListMissing()
    {
        for (var i = List.Count - 1; i > -1; i--)
        {
            if (List[i] == null)
            {
                List.RemoveAt(i);
                tmp = true;
            }
        }
    }
}
```

23.*EventTrigger*

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

public class EventTrigger : MonoBehaviour
{
    //("Dialog") ## (int 1-99) Contoh: "D1"


    public GameObject[] invisibleWall;


    private bool isEventTriggered = false;


    private void OnTriggerEnter(Collider other)
    {
        if (other.tag.Equals("PlayerSelectable") && isEventTriggered == false && !other.isTrigger)
        {
            Debug.Log("Player Have Trigger [" + gameObject.name + "]");
```

```
            isEventTriggered = true;
            Triggered();
        }
    }

    void Triggered()
    {

        //InvisibleWall
        if (invisibleWall.Length > 0)
        {
            foreach (GameObject obj in invisibleWall)
            {
                Destroy(obj);
            }
        }
        Destroy(gameObject);
    }
}
```