PERANCANGAN GAME ARCADE CORGI ADVENTURE UNTUK ANDROID MENGGUNAKAN UNITY

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

Oleh :
Mohammad Izzudin Pamungkas
2012130056

JURUSAN TEKNIK INFORMATIKA
SEKOLAH TINGGI MANAJEMEN INFORMATIKA & KOMPUTER LIKMI
BANDUNG
2016

PERANCANGAN GAME ARCADE CORGI ADVENTURE UNTUK ANDROID MENGGUNAKAN UNITY

Oleh :
Mohammad Izzudin Pamungkas
2012130056

Bandung, 21 Januari 2016
Menyetujui,

Hery Heryanto, S.Kom., M.Kom.	Dhanny Setiawan, S.T., M.T.
Pembimbing	Ketua Jurusan

JURUSAN TEKNIK INFORMATIKA
SEKOLAH TINGGI MANAJEMEN INFORMATIKA & KOMPUTER - LIKMI
BANDUNG
2016

ABSTRAK

Perkembangan game yang cepat dan jenisnya yang terus bertambah serta dukungan perkembangan perangkat android membuat game tidak bisa terlepas dari kehidupan sehari-hari. Game juga sekarang telah menjadi salah satu sarana hiburan untuk melepas stress dan penat. Grafis dan gameplay yang ditawarkan pun beragam tergantung dari jenis yang dimainkan. Jenis game arcade merupakan tipe yang mudah dipahami dan dimainkan sangat cocok untuk menghiangkan stress. Ditambah 2D grafis yang sederhana membuat game ini semakin nyaman dimainkan.

Berdasarkan latar belakang diatas, maka diambilah tema *game arcade platform* dengan grafis 2D. Pembuatan game menggunakan Unity sudah banyak digunakan. Unity menyediakan komponen *scene* untuk tempat pembuatan dan perancangan desain *game* salah satunya menggunakan media penyimpanan *playerperf*. Interaksi yang digunakan antara *player* dengan objek lainnya yaitu menggunakan *collider component* yang ada pada unity. Komponen ini merupakan sebuah jaring tak terlihat yang mengelilingi bentuk *object* dan bertanggung jawab dalam mendeteksi tabrakan dengan benda lain. Tidak hanya mendeteksi interaksi tabrakan, *component* ini juga dapat mendahulukan tabrakan dengan menggunakan *Ray-Casting*. *Ray-Casting* adalah sebuah garis vektor antara dua point dalam ruang 3D yang dapat digunakan dalam mendeteksi *intersection* dengan *collider* pada *game object*. *Ray-Casting* dapat digunakan untuk mengambil informasi yang berguna seperti jarak dari titik awal hingga akhir garis.

Perancangan *game* dilakukan dengan mengaplikasikan Algoritma Finite State Machine dan menggunakan bahasa pemrograman C# yang dimanfaatkan untuk pengkodean agar objek dalam *game* dapat bergerak dan melakukan operasi yang diperlukan. Dalam pengembangan proyek ini metode pengembangan yang digunakan adalah metode prototype dimana proses pembangunan dan testing akan terus dilakukan sampai tercipta game yang sesuai dan dapat dimainkan di platform Android.

KATA PENGANTAR

Alhamdulillahi Rabbil ‘alamiin, segala puji dan syukur kepada Allah SWT atas berkah, rahmat, hidayah, serta segala kemudahan yang selalu diberikan, sehingga atas izin-Nya penulis dapat menyelesaikan Tugas Akhir ini.

Penulis telah melakukan yang terbaik dalam penyusunan Tugas Akhir ini, namun penulis menyadari bahwa dalam penyusunan Tugas Akhir ini masih terdapat banyak kekurangan. Penulis memohon maaf atas segala kekurangan yang ada dalam Tugas Akhir ini. Kiranya di balik semua kekurangan yang ada, Tugas Akhir ini bisa memberikan manfaat kepada seluruh pembaca Tugas akhir ini.

Selama masa pengerjaan Tugas Akhir ini banyak pihak yang telah memberi dukungan, bekal, serta semangat kepada penulis. Penulis mengucapkan banyak terima kasih kepada:

1. Bapak Hery Heryanto, S.Kom., M.Kom. yang telah meluangkan membimbing, mengoreksi, memberi saran, serta meluangkan waktu dan tenaga untuk membantu penulis menyelesaikan Tugas Akhir ini.
2. Bapak Dhanny Setiawan, S.T., M.T. sebagai ketua jurusan Teknik Informatika yang telah banyak membantu proses akademis dan memberi banyak ilmu kepada penulis selama menempuh kuliah di STMIK LIKMI.
3. Dosen-dosen STMIK LIKMI yang telah memberikan ilmu dan pengajaran kepada penulis selama menempuh pendidikan di STMIK LIKMI.
4. Staff-staff kampus yang telah memberi banyak bantuan selama penulis berkuliah di STMIK LIKMI.
5. Regiana Prianggara sahabat sekaligus graphic designer yang telah membantu dan meluangkan waktu untuk penulis dalam menyusun Tugas Akhir ini.
6. Della Anggraeni kekasih yang selalu menemani, mendampingi, mendukung, dan membantu penulis dalam penyusunan Tugas Akhir

7. Teman-teman jurusan Teknik Informatika angkatan 2012 yang selama kurang lebih empat tahun ini berjuang bersama-sama dalam mencapai kelulusan.
8. Teman-teman mahasiswa/i STMIK LIKMI baik yang berbeda jurusan maupun angkatan yang selalu memberikan dukungan dan penghiburan.
9. Pihak lain yang membantu yang tidak dapat disebutkan namanya satu persatu.

Penulis selalu membuka diri untuk setiap saran maupun kritik membangun yang ditujukan baik penulis maupun isi dari Tugas Akhir ini. Sehingga di lain kesempatan penulis dapat melakukan yang lebih baik lagi.

Bandung, 21 Januari 2016

Penulis

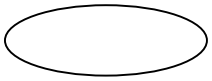
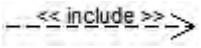
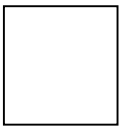
DAFTAR ISI

DAFTAR GAMBAR

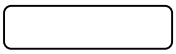
DAFTAR TABEL

DAFTAR SIMBOL

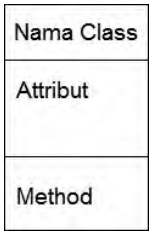
Use case diagram

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
Actor		Entitas eksternal yang berhubungan dengan sistem
Include		Menspesifikasikan bahwa use case sumber secara eksplisit.
System boundary		Menspesifikasikan paket yang menampilkan sistem secara terbatas.

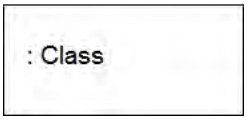
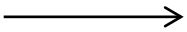
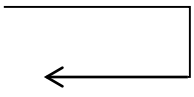
Activity diagram

Nama Simbol	Simbol	Keterangan
Kondisi awal		Menunjukkan kondisi awal kegiatan sistem
Kondisi akhir		Menunjukkan kondisi berakhirnya kegiatan sistem
Action State		Kegiatan yang sistem lakukan
Desicion		Percabangan dimana sistem harus melakukan salah satu cabang keputusan
Transition		Menunjukkan arah sebuah action state ke action state selanjutnya

Class diagram

Nama Simbol	Simbol	Keterangan
Class		Menunjukan nama kelas yang terdapat pada perangkat lunak beserta dengan attribute dan method yang ada di dalamnya.
Directed Association		Relasi antar class dengan makna class yang satu digunakan oleh class yang lain.

Sequence diagram

Nama Simbol	Simbol	Keterangan
Actor		Entitas eksternal yang berhubungan dengan sistem
Object		Menunjukan objek yang berkaitan dengan alur kerja sistem.
Stimulus		Menunjukkan komunikasi antara dua buah objek berupa pengiriman pesan (message).
Self stimulus		Menunjukkan komunikasi berupa pengiriman pesan (message) kepada objek itu sendiri.

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

Pada perkembangan zaman sekarang ini, semakin lama manusia semakin memiliki mobilitas yang tinggi oleh karena itu mereka membutuhkan teknologi yang dapat mendukung mobilitas mereka. *Game Console* dan PC sangat sulit dibawa karena bentuknya yang sangat besar dan sangat besar untuk dibawa kemanapun. Bentuk kecil dan ringan untuk dibawa merupakan salah satu solusi platform yang dapat mendukung mobilitas manusia. Tak jarang bisa ditemukan beberapa *game* komputer yang memiliki requirement yang tinggi. Hal ini yang mendorong beberapa *gamer* beralih ke platform *Android* dalam menyalurkan hobi.

Pengguna *Android* yang semakin banyak mendorong para *developer* untuk mengembangkan *game* mereka di platform ini. Dari *game* bergenre *role playing game*, *adventure*, *puzzle*, simulasi dan masih banyak lagi. Dari berbagai jenis game yang terdapat di Android penulis tertarik untuk membuat *game platform arcade*. Dimana *player* diharuskan untuk menyelamatkan diri dan *player* juga bisa mendapatkan perlengkapan sesuai dengan level yang ada didalam dari *game* ini.

Salah satu game *platform arcade* yang cukup terkenal adalah Rayman Legends buatan Ubisoft Studio. *Gameplay* permainan ini yaitu player sebagai tokoh Rayman hingga empat *player* secara bersamaan membuat jalan mereka melalui berbagai *level*. *Lums* juga dapat dikumpulkan dengan menyentuh, mengalahkan musuh, atau membebaskan *Teensies*.

Dari segi grafis, *gameplay* dan fitur permainan pembanding penulis ingin membuat *game platform arcade* yang lebih baik dan berbeda. Pertama dari segi grafis *game* ini masih 2D yang memungkinkan untuk peningkatan grafis. Kedua dari segi *gameplay* game ini memberikan beberapa *gameplay* yang berbeda, namun masih memungkinkan ditambahkan *level* lain yang lebih menantang. Jadi berdasarkan perbandingan tersebut

dipilihlah topik “Perancangan *Game Platform Arcade Corgi* Adventure untuk Android Menggunakan Unity”.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah diatas, maka maksud dari penulisan skripsi ini adalah sebagai berikut :

1. Bagaimana cara pembuatan *game Platform Arcade* melalui bahasa pemrograman C# untuk perangkat *Android* menggunakan Unity?
2. Bagaimana cara agar *player* dapat berinteraksi dengan *environment* yang ada pada game?

1.3 Tujuan Penulisan

Pembuatan game ini dan penulisan Tugas Akhir ditujukan untuk memenuhi salah satu prasyarat kelulusan pendidikan sarjana (S1) Teknik Informatika di Sekolah Tinggi Manajemen Informatika dan Komputer LIKMI. Pembuatan *game* dan Tugas Akhir ini juga ditujukan untuk mempraktekan ilmu yang telah diperoleh selama masa kuliah juga menguji kemampuan penulis dalam bidang *games programming*.

1.4 Batasan Masalah

Demi memenuhi penulisan dan penyelesaian *game* di waktu yang terbatas ini, maka diperlukan pembatasan masalah dalam penyusunan tugas akhir ini. Batasan tersebut diantaranya :

1. Game *arcade* yang dibuat dan dibahas dalam penulisan tugas akhir ini bersifat *offline*.
2. Game bersifat *single player*.
3. Game hanya bisa dijalankan pada *Android*
4. Game akan terdiri 5 level.

1.3 Kegunaan Hasil

Adapun hasil dari penyusunan tugas akhir ini diantaranya:

1. Menjadi referensi dalam memilih game di *platform* Android.
2. Menghilangkan kejenuhan setelah seharian beraktifitas.
3. Penulisan tugas akhir ini juga diharapkan dapat membantu memahami konsep *games programming* dan menjadi referensi dalam pembuatan tugas akhir di lain waktu.

1.4 Metode Penelitian

Dalam penyusunan tugas ini metode yang digunakan diantaranya :

1. Studi Pustaka

   Penelitian dilakukan dengan membaca dan mengumpulkan dasar – dasar teori dari buku, artikel, website yang berhubungan dengan masalah yang diteliti dan dapat mendukung penelitian dan pembuatan game ini.

2. Analisis

   Setelah teori yang dikumpulkan dinilai cukup dilanjutkan dengan menganalisis objek dan komponen yang diperlukan dalam pembangunan game ini.

3. Perancangan

   Merancang desain awal perangkat lunak ini dan gambaran umum agar terarah sesuai dengan hasil analisis objek dan komponen yang dilakukan sebelumnya.

4. Implementasi

   Mengimplementasikan hasil perancangan perangkat lunak menjadi kode program dengan menggunakan bahasa pemrograman.

5. Pengujian

   Setelah perangkat lunak selesai di kodekan dilakukan pengujian untuk mencari error maupun bug yang mungkin terdapat pada perangkat lunak.

1.5 Sistematika Penulisan

Sistematika penulisan Skripsi ini disusun untuk memberikan gambaran umum tentang game yang dibuat. Sistematika penulisan tugas akhir ini adalah sebagai berikut :

BAB I : PENDAHULUAN

Menguraikan tentang latar belakang permasalahan, identifikasi masalah, tujuan dibuatnya game, batasan masalah, kegunaan hasil, metode penelitian serta sistematika penulisan.

BAB II : TINJAUAN PUSTAKA

Bab ini menjelaskan teori tentang rekayasa perangkat lunak, teori object oriented methodology, teori game dan teori algoritma finite state machine yang dipakai penulis untuk mendukung penulisan Tugas Akhir.

BAB III : ANALISIS DAN PERANCANGAN SISTEM

Berisi gambaran umum dari game yang akan dibuat, analisa spesifikasi kebutuhan perangkat lunak, dan pemodelan game menggunakan use case diagram, activivity diagram, class diagram dan sequence diagram. Analisa dilakukan berdasarkan konsep dan teori yang dipakai.

BAB IV : IMPLEMENTASI DAN PENGUJIAN SISTEM

Dalam bab ini penulis menjelaskan implementasi game yang dikembangkan dari hasil analisis sebelumnya serta melakukan pengujian pada navigasi tombol-tombol yang ada, fungsi game, serta kompatibilitas pada perangkat Android.

BAB V : KESIMPULAN DAN SARAN

Merupakan Merupakan bab terakhir yang berisi tentang kesimpulan dari identifikasi masalah yang ada juga hasil yang diperoleh dari pengembangan aplikasi yang sudah dilakukan. Bab ini juga berisi saran yang dapat digunakan sebagai acuan dalam mengembangkan game ini maupun game sejenis di masa yang akan datang.

BAB II

LANDASAN TEORI

2.1 Rekayasa Perangkat Lunak

Dalam pembangunan perangkat lunak sering kita mendengar istilah rekayasa perangkat lunak. Berikut beberapa pengertian rekayasa perangkat lunak menurut para ahli di bidang perangkat lunak diantaranya:

Rekayasa perangkat lunak menurut Fritz Bauer yang terdapat dalam buku "Software Engineering: A Practitioner's Approach 7th Edition" adalah sebagai berikut:

"*Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines*." (Pressman, 2010:13).

Menurut Ivan Marsic dalam bukunya Software Engineering, rekayasa perangkat lunak didefinisikan sebagai :

"*Software engineering is a discipline for solving business problems by designing and developing software-based systems*." (Marsic, 2012:1).

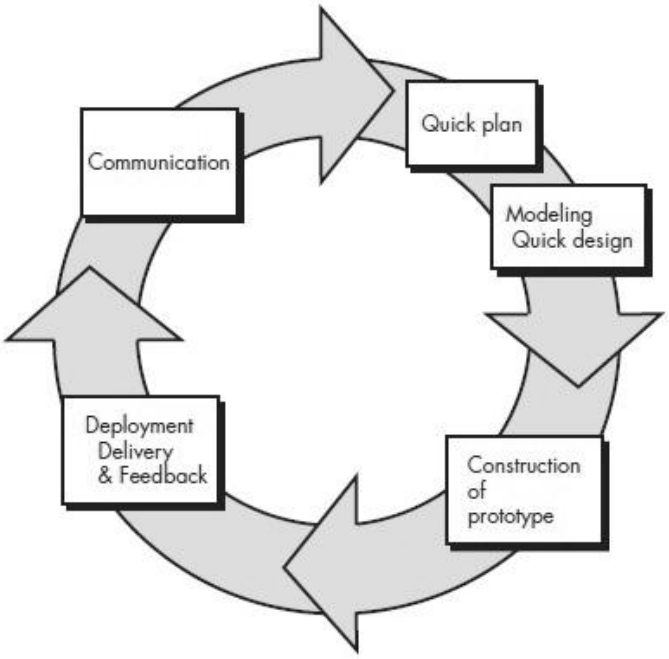
Menurut Bernd Bruegge dan Allen H. Dutoit dalam buku Object – Oriented Software Engineering 2nd Edition rekayasa perangkat lunak diartikan sebagai:

"*Software engineering is a modelling activity. Software engineers deal with complexity through modelling, by focusing at any one time only the relevant details and ignoring eeverything else*."(Bruegge, 2004:5).

Dari beberapa pengertian tentang rekayasa perangkat lunak menurut para ahli di atas, penulis menyimpulkan bahwa rekayasa perangkat lunak adalah disiplin ilmu komputer yang memiliki aturan yang sistematis dan diterapkan oleh perekayasa perangkat lunak untuk mendesain, membangun, dan memelihara suatu perangkat lunak yang dapat berfungsi baik pada komputer sesuai dengan kebutuhan dari penggunanya.

2.2 Metode Pengembangan Prototype

Metode ini merupakan salah satu dari metode yang ada pada *Software Development Life Cycle*. Berbeda dengan model pengembangan *waterfall* yang memiliki tahap yang panjang sebelum produk perangkat lunak dihasilkan, metode *prototyping* mengembangkan perangkat lunak melalui perbaikan terus-menerus dan komunikasi antara *client* dan *developer* yang berawal dari dibuatnya model perangkat lunak dasar (*prototype*).



Gambar 2.1
Model *Prototype*
Sumber : *Sottware Engineering : A Practitioner's Approach 7th Edition*
Roger Pressman (2010:43**)**

1. Pengumpulan Kebutuhan

   Tahapan ini pelanggan dan *developer* bersama-sama mendefinisikan format seluruh perangkat lunak, mengidentifikasikan semua kebutuhan yang diinginkan, dan garis besar sistem yang akan dibuat.

2. Membangun *Prototype*

   Tahapan ini *developer* membuat perancangan sementara yang berfokus pada penyajian kepada pelanggan, misalnya dengan membuat format input dan format output dari perangkat lunak yang akan dibuat oleh *developer*.

3. Evaluasi *Protoptyping*

   Evaluasi dilakukan oleh pelanggan apakah *prototype* perangkat lunak yang sudah dibuat oleh *developer* sudah sesuai dengan keinginan pelanggan atau belum. Jika sudah sesuai maka akan dilanjutkan ke tahapan berikutnya. Jika tidak *prototyping* direvisi dengan mengulangi tahap 1, 2, dan 3.

4. Mengkodekan Sistem

   Dalam tahap ini *prototype* perangkat lunak yang sudah di sepakati oleh pelanggan dan *developer* akan diterjemahkan ke dalam bahasa pemrograman yang sesuai oleh *programmer*.

5. Menguji Sistem

   Setelah sistem *prototyping* sudah menjadi suatu perangkat lunak yang siap pakai, harus dites dahulu sebelum digunakan. Pengujian ini dilakukan dengan *White Box*, *Black Box*, pengujian arsitektur dan lain-lain. Jika ditemukan error atau bug programmer akan melakukan perbaikan pada perangkat lunak.

6. Evaluasi Sistem

   Pelanggan akan mengevaluasi apakah *software* yang sudah diserahkan *developer* sudah sesuai dengan yang diharapkan atau belum. Jika ya maka akan dilakukan ke tahapan yang ke-7. Jika tidak, maka *developer* akan mengulangi tahapan 4 dan 5.

7. Menggunakan Sistem

   Ini merupakan tahapan terakhir dan final dari model *prototyping* dimana *software* yang telah dibuat dan diuji *developer* dan diterima pelanggan siap untuk digunakan.

Dalam pengembangan game yang dilakukan tidak ada pihak pelanggan maupun pihak *developer*. Maka pembangunan *game* ini akan dilakukan secara terus – menerus sampai dicapai kondisi yang dianggap sudah memenuhi kebutuhan dan rencana awal.

2.3 Object Oriented Methodology (OOM)

Object-Oriented Methodology (OOM) atau metodologi berorientasi objek adalah pendekatan pengembangan sistem baru yang mendorong dan memfasilitasi penggunaan kembali komponen perangkat lunak. Dengan metodologi ini, sistem komputer bisa

dikembangan secara per komponen yang memungkinkan penggunaan kembali komponen efektif yang sudah ada dan memfasilitasi pembagian komponennya dengan sistem lain. Tujuan utama dari metodologi berorientasi objek adalah perakitan dan pembangunan perangkat lunak dari komponen yang ada.

Penggunaan metodologi berorientasi objek mulai diperkenalkan sekitar tahun 1990 dengan peluncuran metodologi dari Ivar Jacobson, Grady Booch, dan James Rumbaugh. Pada tahun 1989 *Object Management Group* (OMG) didirikan dengan misi untuk menetapkan pedoman *industri*, spesifikasi manajemen objek yang rinci dan kerangka kerja umum untuk pengembangan aplikasi. Salah satu spesifikasi yang terkenal dan dihasilkan oleh OMG adalah *Unified Modelling Language* (UML). UML merupakan bahasa untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan sistem perangkat lunak, serta untuk pemodelan bisnis dan sitem non-perangkat lunak lainnya. Standar ini kemudian diadopsi dalam OOM dari OGCIO pada tahun 1999. Sebuah proyek percontohan dilakukan pada bulan Juli 1999 dan kemudian tiga proyek percontohan lainnya diluncurkan antara tahun 2000-2001. OOM itu digulirkan di OGCIO (kemudian ITSD) pada bulan Februari 2002 dan menjadi salah satu standar untuk pengembangan sistem. (OGCIO, 2009)

Dalam penggunaan metodologi berorientasi objek ada beberapa manfaat yang didapatkan, yaitu (OGCIO, 2009):

1. Meningkatkan produktivitas

   Pengembangan aplikasi yang difasilitasi oleh penggunaan kembali komponen yang ada dapat meningkatkan produktivitas dan memfasilitasi pengiriman cepat.

2. Menghasilkan sistem bermutu tinggi

   Kualitas sistem dapat ditingkatkan karena dibangun dengan secara per komponen dengan penggunaan komponen yang ada dan sudah diuji dengan baik dan sudah terbukti.

3. Biaya pemeliharaan lebih rendah

Properti terkait pada OOM dapat ditelusuri dan dapat membantu untuk memastikan dampak perubahan terlokalisir dan masalah yang terjadi dapat dengan mudah ditelusuri. Akibatnya, biaya pemeliharaan dapat dikurangi.

4. Memfasilitasi re-use

   Dengan pendekatan ini, sistem komputer dapat dikembangkan secara per komponen yang memungkinkan penggunaan kembali komponen efektif yang ada. Peluang untuk penggunaan kembali difasilitasi oleh akumulasi dan pengelolaan yang baik terhadap komponen reusable baik yang dikembangkan secara internal atau diperoleh secara eksternal.

5. Manajemen kompleksitas

   Penggunaan OOM memudahkan proses dalam mengelola kompleksitas. Dengan diuraikannya dari solusi yang kompleks menjadi komponen-komponen yang berbeda dan dengan setiap komponen dikemas (misalnya diperlakukan sebagai kotak hitam) dari yang lainnya, pengembangan yang kompleks dapat dikelola dengan lebih baik.

2.3.1 Object Oriented Analyst (OOA)

Object-Oriented Analysis (OOA) merupakan bagian dari metodologi berorientasi objek, pada tahap ini akan dianalisa objek dan kelas apa saja yang dibutuhkan dalam proses pembangunan perangkat lunak. Adapun beberapa pendapat dari para ahli mengenai object-oriented analysis. Menurut Grady Booch dalam buku "*Object – Oriented Analysis and Design with Applications 3rd Edition*", Grady Booch dikatakan bahwa:

"*Object-Oriented Analysis a method of analysis in which requirements are examined from the perspective of the classes and objects found in the vocabulary of the problem domain*." (Booch, 2007:598)

Menurut Jeffrey L. Whitten dan Lonnie D. Bentley dalam buku "*System Analysis and Design Methods 7th Edition*" dikatakan bahwa:

"*Object-Oriented Analysis (OOA) an approach used to (1) study existing objects to see if they can be reused or adapted for new uses and (2) define new or modified objects that will be combined with existing objects into a useful business computing application*." (Whitten dan Bentley, 2007:370)

Dari pengertian yang ada dapat disimpulakan OOA adalah metode analisis di mana kebutuhan diperiksa dari sudut pandang kelas dan objek. OOA juga digunakan untuk mempelajari objek yang sudah ada apakah dapat digunakan lagi dan membuat objek baru yang akan dikombinasikan dengan objek yang sudah ada menjadi perangkat lunak.

2.3.2 Object Oriented Design (OOD)

Object-Oriented Design (OOD) merupakan tahap lanjut dari *object-oriented analysis*. Berikut definisi *object-oriented design* menurut para ahli. Menurut Grady Booch dalam buku "*Object – Oriented Analysis and Design with Applications 3rd Edition*", Grady Booch dikatakan bahwa:

Object-Oriented Design a method of design encompassing the process of objectoriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design. Specifically, examples of this notation include class diagrams, object diagrams, component diagrams, and deployment diagrams. (Booch, 2007:598)

Menurut Jeffrey L. Whitten dan Lonnie D. Bentley dalam buku "*System Analysis and Design Methods 7th Edition*" dikatakan bahwa:

"*Object-oriented design (OOD), a new design strategy that follows up object-oriented analysis to refine object requirement definitions and to define new design-specific objects*." (Whitten dan Bentley, 2007:468)

Dari pengertian yang ada dapat disimpulkan bahwa OOD adalah metode desain yang meliputi proses dekomposisi *object oriented* dan notasi untuk menggambarkan model baik logis dan fisik serta statis dan dinamis dari sistem. Secara khusus OOD meliputi *class diagram, object diagram, component diagram*, dan *deployment diagram*.

2.3.3 Unified Model Language

Unified Model Language (UML) adalah bahasa pemodelan visual yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan sistem sofware diantaranya menangkap keputusan dan pemahaman tentang sistem yang harus dibangun. UML biasanya digunakan untuk memahami desain, browsing, mengkonfigurasi,

memelihara, dan mengendalikan informasi tentang sistem tersebut. hal ini dimaksudkan untuk digunakan dengan semua metode pengembangan, tahap siklus hidup, domain aplikasi, dan media. (Raumbaugh, James.2004:3)

Dalam UML terdapat beberapa diagram yang dikenal, yaitu (Whitten dan Bentley, 2007:246):

1. *Use Case Diagram*

   Use Case Diagram digunakan untuk menggambarkan fungsi dari sistem yang akan di buat dari sudut pandang pengguna eksternal, dari materi dan terminologi yang mereka pahami. Dalam penyelesaian permintaan yang membutuhkan interaksi dengan pengguna atau ahli secara akurat dan sepenuhnya.

Dalam Use Case Diagram ini terbagi lagi menjadi dua komponen penyusun, yaitu:

a. *Actor*

   Actor adalah pelaku yang memulai aktivitas sistem dengan tujuan menyelesaikan beberapa tugas bisnis yang menghasilkan suatu hasil yang menghasilkan nilai dan dapat diukur.

   Secara umum actor dapat dibagi menjadi 4 kelompok, yaitu:

   (1) *Primary Bussiness Actor* merupakan *actor* yang menerima keuntungan utama dari pengeksekusian *Use Case* dengan menerima sesuatu yang bernilai, contohnya pegawai yang menerima gaji setiap bulan dari sistem penggajian.

   (2) *Primary System Actor* merupakan *actor* yang terlibat langsung dengan sistem untuk memulai atau memicu proses bisnis atau sistem, contohnya kasir yang memindai kode barang saat transaksi dengan pelanggan.

   (3) *External Server Actor* merupakan *actor* yang bertanggung jawab merespon permintaan dari *use case*, contohnya validasi oleh bank saat pengunaan kartu kredit.

   (4) *External Receiver Actor* merupakan *actor* yang bukan merupakan *actor* utama tetapi menerima sesuatu yang dapat diamati atau diukur dari nilai output dari *use case*, contohnya bagian gudang menerima surat perintah untuk mempersiapkan pengiriman setelah pelanggan telah melakukan pemesanan.

b. *Relationships*

Relationships adalah hubungan yang digambarkan sebagai garis antara dua simbol di dalam *use case*. Arti setiap hubungan mungkin berbeda tergantung pada bagaimana garis ditarik dan apa jenis simbol yang menghubungkan. *Relationships* dalam *use case* terbagi menjadi lima hubungan yang berbeda yaitu:

(1) *Association*

Association merupakan suatu hubungan antara seorang aktor dan sebuah *use case* yang menggambarkan interaksi yang terjadi diantara mereka. Terdapat dua jenis garis yang termasuk *association*, yaitu garis yang memiliki ujung panah dan tidak memiliki ujung panah. Garis yang memiliki ujung panah memberikan arti bahwa *use case* tersebut dapat dilakukan langsung oleh seorang aktor, sedangkan garis yang tidak memiliki panah memiliki arti bahwa terjadi interaksi antara *use case* dan seorang aktor *external server* atau *external receiver*.

(2) *Extends*

Extends merupakan hubungan yang digunakan untuk menyederhanakan *use case* yang kompleks agar lebih mudah dimengerti. Hasil dari penyederhanaan use case tersebut disebut *extension use case* yang berfungsi memperluas fungsionalitas dari *use case* awal. Garis yang menghubungkan *extension use case* dan *use case* inilah yang disebut hubungan *extends*. Garis yang memiliki hubungan ini harus diberi label <<extends>>.

(3) Use atau Includes

Terkadang dapat ditemukan dua atau lebih *use case* menggunakan langkah yang sama. Sebaiknya langkah tersebut dipisahkan dan dibuat menjadi *use case* yang terpisah yang biasa disebut *abstract use case*. *Abstract use case* mewakili sebuah bentuk '*re-use'* yang bisa mengurangi redundansi di dalam *use case*. Garis penghubung antara *abstract use case* dan *use case* inilah yang disebut hubungan *uses/include.* Setiap garis yang memiliki *relationship uses/includes* harus diberi label <<uses>> atau <<includes>>.

(4) *Depends on*

Suatu hubungan antara *use case* yang saling mempengaruhi *use case* lain yang menyebabkan suatu *use case* tidak dapat dijalankan jika *use case* lain belum dijalankan juga. Setiap garis yang memiliki *relationship depends on* harus diberi label <<depends on>>.

(5) Inheritance

Hubungan ini dibuat jika terdapat dua atau lebih aktor yang mempunyai tugas yang sama dan aktor abstrak dibuat untuk menyatukan aktor-aktor tersebut. Garis yang menandakan *relationship inheritance* ini sama seperti *association*, yaitu garis dengan anak panah diujungnya namun yang berbeda pada ujung panah hubungan *inheritance* berwarna putih.

2. *Activity Diagram*

Activity Diagram merupakan salah satu komponen lain dari UML yang berfungsi untuk memodelkan proses dari sistem*. Activity diagram* terlihat sama dengan *flowchart* di mana menggambarkan gambaran aliran proses baik dari kegiatan bisnis maupun *use case*. Hal yang membedakan *activity diagram* dengan *flowchart* adalah *activity diagram* bisa menggambarkan suatu proses yang terjadi secara parallel.

3. *Sequence Diagram*

Sequence diagram dapat dikatakan merupakan bentuk visualisasi dari skenario di *Use Case Specification*, menunjukkan interaksi antar *class*, *actor*, kegiatan apa saja yang dilakukan, urutan kegiatan, informasi yang dibutuhkan untuk masing-masing kegiatan. *Sequence diagram* memiliki peran sangat penting karena akan menjadi pedoman dalam proses pemrograman.

4. *Class Diagram*

Class Diagram adalah diagram yang menggambarkan struktur statis objek suatu sistem, menunjukkan objek kelas yang ada sistem ini terdiri dari serta hubungan antara kelas objek yang ada.

2.3.4 Object Oriented Programming

Pemograman Berbasis Objek atau *Object Oriented Programming* adalah sebuah pendekatan dalam software development berbasis objek yang berinteraksi satu sama lain untuk menyelesaikan sebuah perkerjaan. Interaksi ini berbentuk pesan yang disalurkan antara objek secara berkala. Dalam me-respon pesan, objek dapat melakukan sebuah aksi. (Dan, Clark.2013:7)

Dalam pemrograman berorientasi objek terdapat karakteristik yang membedakannya dengan metode terstruktur. Karakteristik itu diantaranya :

1. Polimorfisme/*Polymorphism*

   Polimorfisme menurut Alan Dennis, Barbara Haley Wixom, dan David Tegarden dalam buku "System Analysis Design UML Version 2.0 An Object-Oriented Approach 3rd Edition" dikatakan bahwa:

   "*Polymorphism means having the ability to take several forms. By supporting polymorphism, object-oriented systems can send the same message to a set of objects, which can be interpreted differently by different classes of objects."* (Dennis, Wixon, dan Tegarden, 2009:321)

   Polimorfisme menurut Graddy Booch dalam buku "Object – Oriented Analysis and Design with Applications 3rd Edition" dikatakan bahwa:

   Polymorphism is a concept in type theory, according to which a name (such as a variable declaration) may denote objects of many different classes that are related by some common superclass; thus, any object denoted by this name can respond to some common set of operations in different ways. (Booch, 2007:599)

   Dari pengertian para ahli dapat disimpulkan bahwa polimorfisme adalah kemampuan objek untuk memmiliki banyak bentuk dan menghasilkan hasil akhir yang sama dengan melakukan proses operasi yang berbeda.

2. Enkapsulasi/*Encapsulation*

   Enkapsulasi menurut Alan Dennis, Barbara Haley Wixom, dan David Tegarden dalam buku "*System Analysis Design UML Version 2.0 An Object-Oriented Approach 3rd Edition*" dikatakan bahwa:

"Encapsulation is the mechanism that combines the processes and data into a single object." (Dennis, Wixon, dan Tegarden, 2009:321)

Enkapsulasi menurut Graddy Booch dalam buku "*Object – Oriented Analysis and Design with Applications 3rd Edition*" dikatakan bahwa:

> "Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior. Encapsulation separates the contractual interface of an abstraction and its implementation." (Booch, 2007:594)

Dari pengertian yang ada dapat disimpulkan bahwa enkapsulasi adalah mekanisme penggabungan proses dan data menjadi objek tunggal dan menggolongkan elemen abstraksi yang merupakan stuktur dan perilakunya.

3. Turunan/Inheritance

Turunan menurut Alan Dennis, Barbara Haley Wixom, dan David Tegarden dalam buku "System Analysis Design UML Version 2.0 An Object-Oriented Approach 3rd Edition" dikatakan bahwa:

> *Inheritance allows developers to define classes incrementally by reusing classes defined previously as the basis for new classes. Although we could define each class separately, it might be simpler to define one general superclass that contains the data and methods needed by the subclasses and then have these classes inherit the properties of the superclass.* (Dennis, Wixon, dan Tegarden, 2009:322)

Turunan menurut Graddy Booch dalam buku "Object – Oriented Analysis and Design with Applications 3rd Edition" dikatakan bahwa:

> *Inheritance is a relationship among classes, wherein one class shares the structure or behavior defined in one (single inheritance) or more (multiple inheritance) other classes. Inheritance defines an "is a" hierarchy among classes in which a subclass inherits from one or more generalized superclasses; a subclass typically specializes its superclasses by augmenting or redefining existing structure and behavior.* (Booch, 2007:595)

Dari pengertian para ahli dapat disimpulkan bahwa turunan adalah hubungan antar kelas, di mana satu kelas atau lebih diturunkan menjadi subkelas yang memiliki method dan atribut seperti pada supekelasnya.

Dalam pemrograman berorientasi objek juga terdapat istilah istilah sebagai berikut:

1. Class

Kelas adalah suatu kumpulan objek yang berbagi struktur umum dan perilaku umum. Istilah kelas dan tipe biasanya interchangeable atau dapat saling dipertukarkan. Kelas juga merupakan konsep yang sedikit berbeda dari tipe karena kelas menekankan pada klasifikasi struktur dan perilaku. Kelas memiliki tiga area pokok yaitu nama, atribut dan metoda. Atribut dan metoda dapat memiliki salah satu sifat berikut :

a. *Public*, metoda dan atribut dapat dipanggil oleh kelas manapun.
b. *Private*, metoda dan atribut tidak dapat dipanggil dari luar kelas tersebut.
c. *Protected*, metoda dan atribut hanya bisa dipanggil dari kelas kelas yang bersangkutan dan subkelas turunannya.

2. Object

Objek adalah sesuatu yang dapat melakukan sesuatu hal. Sebuah objek memiliki keadaan/*state*, perilaku/*behavior*, dan identitas/*identity* dari objek serupa yang didefinisikan di dalam kelas mereka.

3. Attribut

Atribut adalah bagian dari kelas yang memiliki nilai dan berkontribusi terhadap definisi keadaan dari kelas.

4. Method

Method adalah operasi pada objek, yang didefinisikan sebagai bagian dari deklarasi kelas. Semua metode adalah operasi, tetapi tidak semua operasi adalah metode. Dalam beberapa bahasa, metode berdiri sendiri dan dapat didefinisikan ulang di *subclass*.

5. Constructor/Konstruktor

Konstruktor merupakan suatu fungsi yang didefinisikan di dalam kelas, konstruktor harus mempunyai nama sama dengan nama fungsinya, dan dijalankan bersamaan dengan terciptanya kelas tersebut. Di dalam suatu kelas bisa terdapat lebih dari satu konstraktor. Konstraktor mirip seperti method tetapi tidak mengembalikan nilai.

6. *DeConstructor*/ De-Konstruktor

De-Konstruktor merupakan suatu fungsi yang didefinisikan di dalam kelas, memiliki nama sama dengan nama fungsinya, tetapi dijalankan bersamaan dengan dimusnahkan atau ditutupnya kelas tersebut.

7. *Properties*/Properti

   Properti merupakan ciri yang dimiliki oleh suatu objek, karakteristik ini juga sebagai pembeda objek satu dengan objek yang lainnya dalam kelas yang sama

2.4. Game

2.4.1 Pengertian Game

Game menurut David Parlett yang terdapat pada buku “Rules of Play - Game Design Fundamentals” karangan Katie Salen dan Eric Zimmerman dikatakan bahwa :

“A formal game has a twofold structure based on ends and means:

Ends. It is a contest to achieve an objective (The Greek for game is agôn, meaning contest.) Only one of the contenders, be they individuals or teams, can achieve it, since achieving it ends the game. To achieve that object is to win. Hence a formal game, by definition, has a winner; and winning is the "end" of the game in both senses of the word, as termination and as object.

Means. It has an agreed set of equipment and of procedural "rules" by which the equipment is manipulated to produce a winning situation. (Zimmerman, 2004:Ch.7:4)

Menurut Greg Costikyan yang terdapat pada buku “*Rules of Play - Game Design Fundamentals*” karangan Katie Salen dan Eric Zimmerman dikatakan bahwa:

“*A game is a form of art in which participants, termed players, make decisions in order to manage resources through game tokens in the pursuit of a goal*.” (Zimmerman, 2004:Ch.7:8)

Menurut Katie Salen dan Eric Zimmerman yang terdapat pada buku Rules of Play-Game Design Fundamentals, game didefinisikan sebagai:

“*A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome*.” (Zimmerman, 2004:Ch.7:11)

Dari pengertian di atas penulis menyimpulkan bahwa game adalah suatu sistem di mana pemain sebagai pengambil keputusan mengelola sumber daya dalam permainan untuk mengejar tujuan dan dalam game pemain dihadapkan dengan konflik buatan dan batasan/aturan.

2.4.2 Arcade Game

Arcade game adalah *genre game* yang merujuk pada permainan mesin *arcade* yang dioperasikan menggunakan koin. Mesin ini sangat terkenal pada era 1970 – 1980. *Arcade game* sudah dikenal satu dekade sebelum *video game* ditemukan. Cikal bakal dari *arcade game* adalah *pinball* yang pengopersiannya masih manual tanpa menggunakan listrik maupun elemen elektronik.

Di tahun 1971, Nolan Bushnell bersama dengan Ted Dabney menciptakan *game* ber-*genre arcade* yang pertama bertemakan Spacewar. Tahun 1972, Nolan dan Ted memulai Atari komputer. Kemudian dia mengembangkan game berjudul Pong yaitu *game* pertama yang tersedia untuk publik atau dijual secara bebas. Karena *game-game* sebelumnya hanya ada di dalam komputer *mainframe* untuk kesenangan sendiri saja. Asal usul Pong dimulai saat Nolan ingin membuat *game* sederhana dan mudah dimengerti. Dengan *memory* dan *micro processor* kelas rendah, kemampuan proses yang terbatas dan grafis yang sederhana, akhirnya dia membuat versi elektronik dari permainan ping pong yang kemudian menjadi Pong yang menjadi cikal bakal *video game* ber-*genre arcade* sampai saat ini.

Menurut Richard Rouse dalam bukunya *Game Design: Theory & Practice game* arcade memiliki ciri sebagai berikut (Rouse,2005:59):

1. *Single screen play*, sebagian besar permainan hanya terjadi pada satu layar saja dan pergerakan *player* dibatasi hanya di sekitar layar saja, bahkan terkadang hanya menggunakan setengah layar.
2. *Infinite play*, *player* bisa bermain game untuk selamanya. *Game* tidak akan tamat atau berakhir, terkecuali *player* kehabisan waktu atau lives. Hal ini dilakukan untuk

memungkinkan *player* untuk melihat berapa lama mereka bisa bermain dan apakah *player* bisa mengalahkan rekornya sendiri.

3. *Multiple lives*, game menyediakan beberapa kesempatan atau nyawa untuk *player* mencoba lagi dan menganalisis pola permainan.
4. *Scoring/high score*, hampir semua *game arcade* memiliki fitur penilaian atau *scoring*, di mana *player* akan mengumpulkan poin untuk mencapai tujuan yang berbeda dalam permainan.
5. *Easy to learn*, *gameplay game arcade* sangat mudah dimengerti dan dimainkan, hanya dengan sekali mencoba bermain *player* sudah menguasai cara bermain *game* ini.
6. *No story*, semua *game arcade* tidak memiliki jalan cerita ataupun cerita yang disampaikan pada *player*. Ciri ini yang paling membedakan game ini dari *game* lain.

Dari ciri-ciri yang telah disebutkan dapat disimpulkan bahwa *arcade game* adalah *game* yang dibuat untuk tujuan "*just for fun*" karena mudah untuk dimainkan dan diperuntukan untuk kejar-mengejar skor tertinggi tanpa memiliki alur cerita maupun cerita yang ingin disampaikan. *Player* juga hanya dituntut untuk memiliki refleks yang baik dalam memainkannya.

2.5. Algoritma Finite State Machine

Algoritma *Finite State Machine* atau biasa disebut sebagai algoritma FSM merupakan salah satu metode penerapan dan pembuatan *artificial intelligence* atau AI. Menurut Ferdinand Wagner dkk dalam buku "*Modeling Software with Finite State Machines: a Practical Approach*", disebutkan bahwa:

The finite state machine introduces a concept of a state as information about its past history. All states represent all possible situations in which the state machine may ever be. Hence, it contains a kind of memory: how the state machine can have reached the present situation. As the application runs the state changes from time to time, and outputs may depend on the current state as well as on the inputs. (Wagner,2006:64)

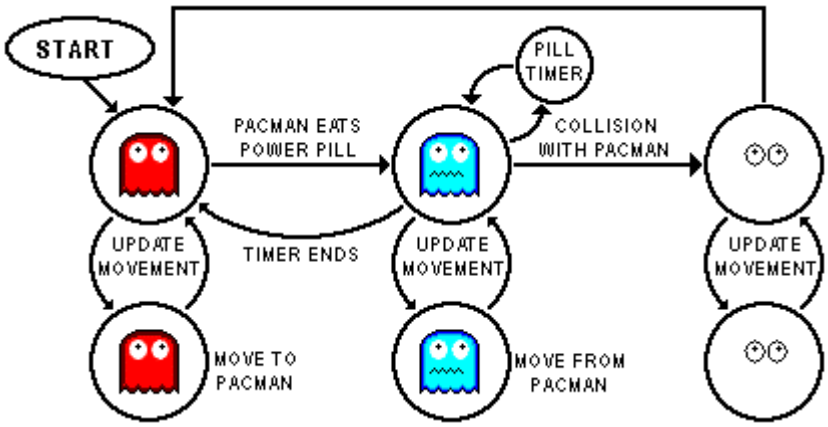
Dapat disimpulkan bahwa *finite state machine* adalah alat atau model dari sebuah perangkat yang memiliki keadaan atau state yang terbatas yang berisi informasi yang merepresentasikan situasi yang mungkin dilakukan. Antar *state* akan terjadi transisi yang akan menghasilkan *output* berupa aksi yang dilakukan. Beberapa jenis aksi dapat

dibedakan tergantung pada kondisi dan saat mereka dilakukan, aksi tersebut tebagi menjadi (Niel,2011:54):

1. *Entry Method*, merupakan metode yang dilakukan pada saat *state machine* memasuki state.
2. *Exit Method*, merupakan metode yang dilakukan pada saat *state machine* keluar dari state.
3. *Input Method*, merupakan metode yang dilakukan ketika kodisi masukan atau *input* bernilai benar. Masing-masing *state* memiliki *input action* sendiri.
4. *Transition-check Method*, merupakan metode yang dilakukan data terjadi perubahan *state* atau pada saat *state transition*.

Algoritma FSM biasanya digunakan dalam pembuatan AI *game* sederhana maupun rumit. Algoritma FSM sendiri dalam pengimplementasiannya cukup sederhana karena hanya menggunakan perintah *if* atau *case*/*switch* untuk pemicu *state transition*-nya.

Contoh dari pengaplikasian algortma FSM dapat diihat pada Gambar 2.2.



Gambar 2.2
FSM pada Pacman
Sumber :
http://oddwiring.com/archive/websites/mndev/MSB/GD100/fsm_1.gif

Gambar 2.2 menjelaskan FSM dari *Ghost* pada *game* Pacman. State awal terletak pada ghost yang bergerak, jika ghost menemukan pacman dalam jarak tertentu maka *ghost* akan bergerak menuju posisi pacman jika tidak menemukan pacman maka *ghost* akan berjalan secara random untuk mencari *pacman*. Selain itu jika pacman memakan

power pill maka *ghost* akan berubah menjadi biru dan akan berjalan menjauh dari pacman, apabila *timer* atau waktu yang pada *power pill* telah habis maka *ghost* akan kembali ke *state normal* dan apabila *ghost* tersebut bertabrakan dengan pacman maka *ghost* akan kembali ke tempat dimana dia respawn dan kembali ke state awal.

Algoritma *Finite State Machine* ini banyak diterapkan dalam pembuatan AI (*Artificial Intelligence*) pada *game* yang digunakan untuk pengambilan keputusan dan membuat NPC dapat merespon saat suatu *event* atau kondisi terjadi. Terdapat empat komponen utama dalam algoritma FSM sederhana yaitu *states*, *state transitions*, *rules*, dan *input events*. *States* adalah suatu keadaan yang mendefinisikan perilaku dan menghasilkan aksi yang dilakukan oleh objek yang bersangkutan. *State transitions* adalah perubahan dari satu *state* ke *state* lainnya yang memungkinkan terjadinya aksi. *Rules* merupakan kondisi atau aturan yang harus terjadi agar memicu perubahan *state* atau memicu *state transition*. *Input events* adalah *input* secara *internal* atau eksternal, yang dapat memicu kondisi/aturan/label dan mengacu ke *state transitions*.

Algoritma FSM banyak digunakan dikarenakan memiliki banyak keunggulan, yaitu (Buckland,2005:44):

1. *Quick and simple to code*, cepat dan sederhana pengkodeannya karena hanya menggunakan perintah *if* atau *case*/*switch*.
2. *Easy to debug*, mudah di-*debug* karena sistem dipisah menjadi beberapa bagian yang mudah dikelola sehingga saat sistem melakukan kesalahan mudah untuk memelusurinya.
3. *Have little computational overhead*, tidak membebani proses komputasi karena FSM mengikuti *hardcoded rules* sehingga hampir tidak menggunakan waktu komputasi *processor* yang berharga.
4. *Intuitive*, FSM dapat dibuat bekerja seolah-olah seperti intuisi manusia.
5. *Flexible*, fleksibel karena dapat dengan mudah disesuaikan dan diperbaiki oleh *programmer* untuk menghasilkan perilaku yang dibutuhkan.

2.6. Android

Android adalah sebuah sistem operasi *mobile* berbasis Linux dan merupakan *platform open source*. Biasanya digunakan untuk *mobile device* salah satunya adalah *smartphone* dan *tablet computers*. Android dikembangkan oleh Open Handset Alliance yang dipimpin oleh Google, dan perusahaan lainnya.

Source Android itu sendiri tersedia dibawah lisensi *free* and *open source software*. Google mempublikasikan semua kode tersebut dibawah naungan Lisensi Apache 2.0 dan sisanya Linux *Kernel Changes*, dibawah naungan lisensi GNU *General Public License* versi 2.

Android, Inc. didirikan di Palo Alto, California pada Oktober 2003 oleh Andy Rubin (*co-founder* Danger), Rich Miner (*co-founder* Wildfire Communications, Inc.), Nick Sears (sekali Vice President di T-Mobile), dan Chris White (*Head Design* dan pengembangan Interface di WebTV). Niat awal perusahaan adalah untuk mengembangkan sebuah sistem operasi canggih untuk kamera digital, ketika disadari bahwa pasar untuk perangkat itu tidak cukup besar, dan dialihkan upaya mereka untuk menghasilkan sistem operasi smartphone yang dapat menyaingi Symbian dan Windows Mobile. Meskipun prestasi masa lalu pendiri dan karyawan awal Android Inc, maka android pun dioperasikan secara diam-diam dan mengungkapkan bahwa android bekerja pada perangkat lunak untuk ponsel.

Pada 17 Agustus 2005, Google memperoleh Android Inc; kunci karyawan Android Inc, termasuk Rubin, Miner, dan White, tinggal di perusahaan setelah dibeli. Tidak banyak yang diketahui tentang Android Inc pada saat itu, tetapi banyak berasumsi bahwa Google merencanakan untuk memasukkan ponsel pasar dengan langkah ini. Di Google, tim yang dipimpin oleh Rubin mengembangkan platform perangkat *mobile* didukung oleh kernel Linux. Google dipasarkan *platform* untuk produsen handset dan operator pada janji menyediakan fleksibel, sistem diupgrade. Google telah berbaris serangkaian komponen perangkat keras dan perangkat lunak partner dan memberi tanda kepada operator bahwa itu terbuka untuk berbagai tingkat kerjasama pada bagian mereka

Dalam perkembangannya, Android melakukan perbaikan dan peningkatan pada fitur yang ada di dalamnya. Dalam setiap versinya Android memiliki nama yang berbeda-

beda dengan penamaan menggunakan nama kudapan. Berikut versi perkembangan Android sampai saat ini:

1. Android 1.0

   Android 1.0 merupakan versi komersial pertama dari Android yang dirilis pada 23 September 2008. Versi ini belum menggunakan nama kudapan dalam penamaanya. Versi ini memiliki fitur-fitur diantaranya Android Market, *web browser* yang mendukung HTML dan XHTML, dukungan terhadap kamera, *wallpaper*, dan pengelompokan ikon aplikasi menjadi satu folder.

2. Android 1.1

   Android 1.1 dirilis pada 9 Februari 2009 yang masih belum menggunakan nama kudapan dalam penamaannya. Fitur tambahan pada versi ini diantaranya kemampuan untuk menambahkan lamppiran pada pesan, informasi yang lebih detail pada map, dan fitur menampilkan/menyembunyikan *dialpad*.

3. Android 1.5 *Cupcake*

   Android 1.5 *Cupcake* merupakan versi ketiga dari sistem operasi Android yang mulai menggunakan nama kudapan dalam penamaannya. Versi ini dirilis pada 30 April 2009 dengan fitur diantaranya dukungan dalam penggunaan *virtual keyboard* pihak *ketiga* (*virtual keyboard* di luar sistem Android), integrasi *home screen* dan *widgets*, penggunaan *folder* pada *home screen*, dukungan terhadap *stereo Bluetooth*, fungsi *copy paste* pada *web browser*, dan fungsi untuk memutar dan merekam *video*.

4. Android 1.6 *Donut*

   Versi keempat dari Android yang diluncurkan pada 15 September 2009. Pada versi ini terdapat fitur utama diantaranya *Quick search box* untuk sistem pencarian yang lebih cepat dan efisien. perbaikan *interface*, integrasi antara kamera dan perekam *video* dengan galeri, peningkatan Android Market menjadi Google Play, indicator penggunaan baterai, dan fungsi *Text to Speech*.

5. Android 2.0 - 2.1 *Éclair*

   Versi kelima dari Android yang diluncurkan pada 26 Oktober 2009. Pada versi ini terdapat fitur utama diantaranya Google Maps Navigator (*Beta version*), perbaikan

browser, penggunaan multi akun, peningkatan kinerja *keyboard*, fungsi pencarian SMS, dan integrasi terhadap *Microsoft Exchage*.

6. Android 2.2 – 2.3 *Froyo*

   Versi ini dirilis pada 20 Mei 2010, memiliki fitur utama diantaranya dukungan terhadap *Adobe Flash*, *hotspot portable*, *keyboard* multi bahasa, perbaikan kecepatan dan performa sistem operasi Android, dan peningkatan integrasi dengan *Microsoft Exchange*.

7. Android 2.3 – 2.3.7 *Gingerbread*

   Versi ketujuh yang dirilis pada 6 Desember 2010. Fitur utama pada versi ini diantaranya Perbaikan *interface*, NFC, dukungan terhadap SIP VOIP, kemampuan *input* teks yang lebih baik dan cepat, dan peningkatan fungsi *copy paste*.

8. Android 3.0 – 3.2 *Honeycomb*

   Versi kedelapan Android yang dikhususkan untuk Tablet PC. Versi ini dirilis pada 22 Februari 2011 dengan fitur utama *interface* yang dirancang khusus untuk *tablet*, *action bar*, peningkatan *multitasking*, update aplikasi Android standar, fitur *copy paste* yang lebih baik, dan desain ulang *keyboard*.

9. Android 4.0 *Ice Cream Sandwich*

   Dirilis pada 19 Oktober 2011 dengan fitur utama diantaranya peningkatan fitur *multitasking*, *facelock*, widget yang *resizeable*, Android *beam*, peningkatan kinerja dan opsi *email*, perbaikan *input* teks dan suara, dan fitur *soft button* sebagai pengganti tombol fisik.

10. Android 4.1 – 4.3 *Jelly Bean*

    Rilis pada 9 Juli 2012 dengan fitur notifikasi yang lebih luas dan kaya fitur, Google Now, *offline voice dictation*, *interface* yang lebih halus, fitur penempatan *widget* yang lebih pintar, multi user pada Tablet PC, *lockscreen widget*, *daydream*, dukungan terhadap OpenGL, dan Bluetooth Smart Ready.

11. Android 4.4 *Kitkat*

    Rilis pada 31 Oktober 2013 dengan tampilan interface yang lebih baik. Fitur lain yang terdapat pada versi ini diantaranya peningkatan performa yang lebih baik untuk

perangkat terdahulu, *wireless printing*, NFC *Host Card Emulation*, *browser* menggunakan chromium *engine*, *screen recording*, dukungan terhadap *virtual machine*, peningkatan fitur *auto focus kamera*, dan peningkatan keamanan dan performa.

2.7. C#

C# adalah sebuah bahasa yang relatif baru dan merupakan turunan dengan nenek moyangnya yang kembali ke C dan C++. Bahasa ini benar-benar berorientasi objek dan dikembangkan oleh Microsoft untuk bekerja dengan .NET *framework*. C# telah dikembangkan dari belakang dan menggabungkan fitur terbaik dari pemrograman lain bahasa sementara membersihkan masalah. C# memiliki sintaks sederhana dari C ++ dan secara umum apabila yang dapat lakukan di C++ juga dapat dilakukan di C#. (Mulchorne, Keiran. 2010:9)

Selama pengembangan *Framework*, *class library* awalnya ditulis menggunakan sistem *compiler* kode disebut *Simple Managed C* (SMC). Pada bulan Januari 1999, Anders Hejlsberg membentuk tim untuk membangun sebuah bahasa baru pada saat yang disebut Cool, yang berdiri untuk "C-like Object Oriented Language". Microsoft telah mempertimbangkan menjaga nama "Cool" sebagai nama akhir bahasa, tetapi memilih untuk tidak melakukannya untuk alasan merek dagang. Pada saat proyek NET mengumumkan pada Juli 2000 *Professional Developers Conference*, bahasa telah berganti nama C #, dan *library class* dan ASP.NET runtime telah *porting* ke C#. Salah satu tujuan menciptakan *framework* dan bahasa C# adalah untuk memperkenalkan konsep-konsep seperti object *oriented*, *type security*, *garbage collection*, dan *structured-exception* handling langsung ke *platform*.

Sejak merilis C #, Microsoft telah terus-menerus berusaha untuk menambahkan fitur tambahan dan perangkat tambahan ke dalam C#. Sebagai contoh, versi 2.0 menambahkan dukungan untuk generics dan versi 3.0 LINQ ditambahkan untuk mengurangi ketidaksesuaian impedansi antara pemrograman dan pemrograman *database* yang digunakan untuk mengolah *data*. Pada C# 5.0 dimasukan dukungan untuk *programming* dalam mengimplementasikan *parallel* dan *asynchronous programming*.

Microsoft menjanjikan pengembangan dan peningkatan terus menerus pada C# agar C# bisa menjadi bahasa pemrograman yang paling banyak digunakan di dunia.

Microsoft juga berkomitmen kepada programmer .NET akan menambahkan alat yang diperlukan untuk meningkatkan produktivitas dan pengalaman pemrograman intuitif. Meskipun program berbasis C# bisa dibuat menggunakan teks *editor*, para pengembang profesional kebanyakan menggunakan *integrated development environment* (IDE) yang lebih mudah digunakan dan meningkatkan produktivitas. Microsoft telah mengembangkan IDE yang luar biasa dalam Visual Studio (VS). Pada Visual Studio diintegrasikan banyak fitur yang membuat pemrograman pada .NET *framework* yang lebih intuitif. Visual Studio 2012 terdapat fitur baru seperti dukungan *debugging* yang lebih baik untuk pemrograman *paralel* dan meningkatkan pengujian. (Clark, 2013:5)

2.8. Unity

Unity adalah sebuah *platform development tool* yang dikembangkan oleh Unity *Technologies* dan digunakan untuk mengembangkan *game* mulai dari *game desktop based*, *console* (PS3, PSVITA, XBOX360, WiiU), *web* hingga *smartphone* (iOS, Android).

Versi pertama dari Unity (1.0.0) dibuat oleh rekan-rekan: David Helgason, Joachim Ante dan Nicholas Francis di Denmark. Produk awal diluncurkan pada tanggal 6 Juni 2005. Tujuannya adalah untuk menciptakan sebuah mesin permainan yang terjangkau dengan alat profesional untuk pengembang game amatir sementara "demokrasi pengembangan *game*" industri. Dengan terinspirasi oleh alur kerja yang mudah, pipa aset sederhana, dan antarmuka drag-and-drop. Ketika awalnya dirilis, Unity tersedia hanya untuk Mac OS X, dan pengembang hanya bisa menyebarkan ciptaan mereka untuk beberapa *platform*. Versi saat ini (4.5.5 sampai tulisan ini) didukung pada Windows dan Mac OS X, dan menawarkan setidaknya selusin target platform.

Unity juga meraih banyak penghargaan mulai dari tahun 2006 yaitu “Penggunaan Terbaik dari Mac OS X Graphic” di Apple WWDC dan sebagai pertama kalinya bahwa *development tools* mencapai tingkat kualitas dan kemudahan dalam penggunaan hingga Unity dapat memenangkan penghargaan ini. Pada tahun 2009 sebagai “One Top Five Game Companies of the Year” oleh Gamasutra, Pada tahun 2012 Unity menempati ranking

#1 untuk survei teknologi *developing game mobile*, daftar paling tinggi dengan 53,1% dari para pengembang menggunakan Unity sebagai *tools* yang sering digunakan dalam *developing game*.

Pemilihan penggunaan Unity dalam penyusunan tugas akhir penulis dipertiimbangkan dari beberapa aspek kelebihan yang dimiiki Unity, diantaranya :

1. Mudah digunakan dalam men*develop* sebuah *game*.
2. Tersedia untuk hampir semua *platform* yang ada pada generasi sekarang (dengan pengecualian untuk Nintendo 3DS).
3. Alur kerja dari sebuah *asset* mudah dikonfigurasikan.
4. Mendukung beberapa bahasa pemograman seperti C# dan *Javascript*.
5. Adanya *Asset Strore* yang dapat membantu dalam tahap *developing*.

2.9. Android SDK

Android *software development kit* (SDK) termasuk seperangkat alat pengembangan. Ini termasuk *debugger*, *library*, sebuah *emulator handset* berdasarkan QEMU, dokumentasi, *code sample*, dan *tutorial*. Platform pengembangan saat ini didukung termasuk komputer yang menjalankan Linux (setiap yang modern Linux desktop distribusi), Mac OS X 10.5.8 atau yang lebih baru, dan Windows XP atau yang lebih baru. Untuk saat ini kita juga dapat mengembangkan *software* Android di Android itu sendiri dengan menggunakan AIDE yang - Android IDE - Java, C ++ dan aplikasi *Editor* Java. *Integrated Development Environment* (IDE) adalah Eclipse menggunakan Android *Development Tools* (ADT) *Plugin*, meskipun IntelliJ IDEA IDE (semua edisi) mendukung penuh pengembangan Android, dan NetBeans IDE juga mendukung pengembangan Android melalui *Plugin*.

Selain itu, pengembang dapat menggunakan *editor* teks untuk mengedit Java dan XML *file*, kemudian menggunakan alat baris perintah (*Java Development Kit*) untuk membuat, membangun dan *debug* aplikasi Android serta pengendalian terpasang perangkat Android (misalnya, memicu reboot, menginstal paket perangkat lunak secara jarak jauh).

Perangkat tambahan untuk SDK Android berjalan seiring dengan perkembangan *platform* Android secara keseluruhan. SDK juga mendukung versi *platform* Android dalam kasus pengembang ingin menargetkan aplikasi mereka di perangkat yang lebih tua. Alat pengembangan komponen *download*, sehingga setelah satu telah di-*download* versi terbaru dan *platform*, *platform* yang lebih tua dan alat-alat juga dapat di*download* untuk pengujian kompatibilitas.

Adapun kelebihan dari Android-SDK diantaranya:

1. Kemudahan untuk merilis aplikasi yang telah dibuat, tidak adanya proses perizinan, distribusi, atau pengembangan biaya atau persetujuan dalam proses rilis.
2. Adanya akses hardware Wi-Fi.
3. Jaringan GSM, EDGE, dan 3G untuk telepon atau transfer *data* yang memungkinkan kita untuk membuat atau menerima panggilan atau SMS dan untuk mengirim atau mengambil *data* di jaringan *mobile*.
4. Adanya fasilitas API untuk menggunakan sensor *hardware*, termasuk *accelerometer* dan kompas.
5. Komprehensif API untuk layanan berbasis lokasi seperti GPS.
6. *Library* yang berguna untuk *Bluetooth*.
7. IPC *message passing*.
8. Kendali kontrol *multimedia hardware*, termasuk pemutaran dan merekam dengan kamera dan mikrofon.
9. Kemampuan untuk mengintegrasikan hasil pencarian aplikasi ke dalam sistem pencarian.
10. Adanya sebuah *browser open source* HTML5 *WebKit* yang terpadu.
11. Fasilitas *hardware-accelerated grafis mobile* yang optimal, termasuk *library* berbasis grafis 2D dan dukungan untuk grafis 3D menggunakan OpenGL ES 2.0.
12. Fasilitas media yang dilengkapi *library* dan media untuk memutar dan merekam berbagai *audio-video* atau gambar.
13. Fasilitas dukungan *Native Google Maps, Geocoding*, dan GPS.

BAB III

ANALISIS DAN PERANCANGAN PERANGKAT LUNAK

3.1 Gambaran Umum Perangkat Lunak

Game yang dirancang dalam Tugas Akhir ini, merupakan game ber-*genre arcade* dengan tema *Platformer* yang selanjutnya akan disebut Corgi Adventure. Dalam game ini *Player* menjadi seorang anjing corgi yang tersesat disebuah planet tidak dikenal untuk mencari Pesawatnya yang hilang disembunyikan oleh robot.

Pada saat *game* ini pertama kali dijalankan, maka pertama kali *Player* akan memasuki *Stage Tutorial*. Didalam *stage* ini *player* akan diberi penjelasan tentang *basic element* yang ada didalam *game* ini mulai dari *item coin*, *item stimpack*, *bonus block*, *trap* dan salah satu musuhnya yaitu *blue robot*. Apabila *player* telah menyelesaikan *stage tutorial* ini dan menemukan bendera maka *player* akan dipindahkan ke *select stage menu*. *Select stage menu* merupakan *menu* pemilihan *stage* yang terdiri dari 5 *level* yang didalamnya terdapat kesusahan di*level* yang berbeda beda.

Dalam *game* ini *player* bermain menggunakan *virtual button* yang terdapat pada layar android, didalamnya terdapat arrow button yaitu;

1. tombol *arrow up* dan *down*, tombol ini akan mengarahkan karakter keatas dan kebawah apabila karakter tidak dalam keadaan apapun maka kamera akan memperlihatkan bagian atas ataupun bawah dari posisi karakter.
2. tombol arrow *left* dan *right*, tombol ini akan menggerakan karakter kekiri ataupun kekanan.

dan juga action button yang didalamnya terdapat fungsi yang berbeda yaitu;

1. tombol *Jump*, tombol ini berfungsi agar karakter yang dimainkan dapat melompat.
2. tombol *Run*, tombol ini berfungsi agar karakter yang dimainkan dapat berlari panjang.
3. tombol *Dash*, tombol ini berfungsi agar karakter yang dimainkan dapat berlari pendek.

4. tombol *Jetpack*, tombol ini berfungsi agar karakter yang dimainkan dapat menggunakan *jetpack*-nya untuk menjangkau level *platform* yang tidak mudah dicapai hanya dengan melompat.
5. tombol *Fire*, tombol ini berfungsi agar karakter yang dimainkan dapat menembakan senjatanya untuk membunuh musuh.

Game ini juga menyediakan fitur *hidden stage* yang hanya dapat diakses apabila *player* telah menyelesaikan semua *level* dan telah mendapatkan *score* lebih dari 10000. Jika *player* telah memiliki persyaratan itu, maka didalam *level select* akan muncul *level* dimana *player* akan melawan bos dari *blue robot* yaitu *Red robot* dengan kapabilitas yang berbeda dengan *blue robot*.

3.2 Analisis Kebutuhan Perangkat Lunak

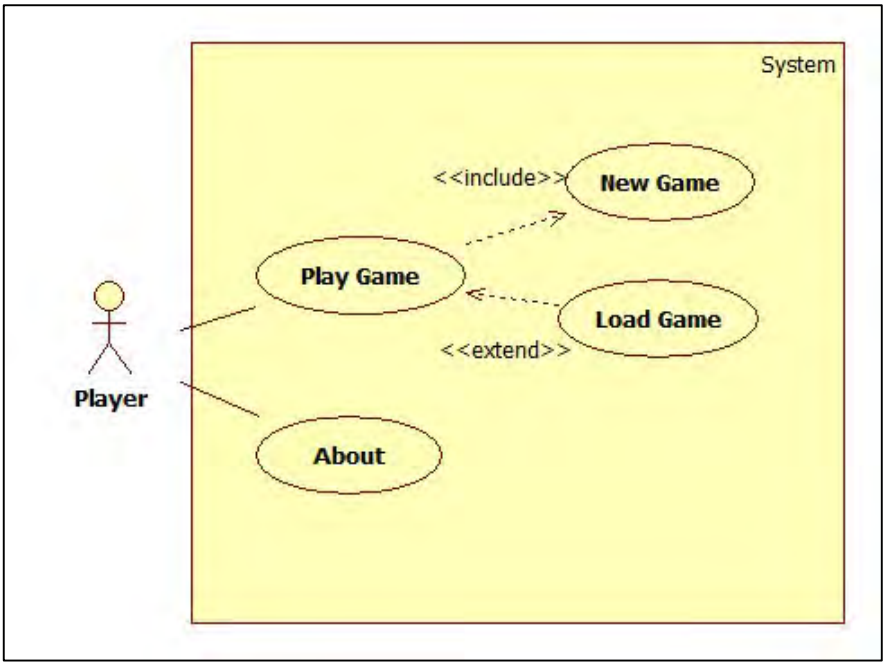
Game Corgi Adventure ini diharapkan dapat memenuhi kebutuhan-kebutuhan dan fitur sebagai berikut:

1. *Game* dapat membuat karakter *player* melakukan suatu aksi didalam permainan sesuai dengan fungsinya berdasarkan dengan perintah yang diberikan *player*.
2. *Game* bersifat *Single-Player*.
3. *Game* harus bisa Pindah *Level*.
4. *Game* harus bisa menggerakan musuh sesuai dengan yang diprogramkan.
5. *Game* harus bisa menampilkan *hidden stage*.
6. *Game* dapat menyimpan data-data player pada *player perf*, baik data *high score* dan banyaknya *coin* yang telah didapatkan.
7. *Game* harus dapat berjalan pada *platform* Android.

3.3 Pemodelan Perangkat Lunak

3.3.1 Use Case Diagram

Pada percanganan game menyediakan 2 menu utama yang ada pada Main Menu yaitu *Play Game* dan *About*. Di dalam menu *Play* Game terdapat 2 pilihan diantaranya *new game*, jika player memilih *new game* maka *player* akan langsung dipindahkan ke *level tutorial* dan apabila *player* telah pernah memainkan *game* ini sebelumnya maka didalam *Play Game menu* akan menampilkan *menu Load Game* maka player akan dipindahkan ke *Level Select*. Didalam game ini fitur *save* akan akan berjalan secara otomatis maka *player* tidak perlu untuk *save game* secara *manual*. Seperti pada Gambar 3.1.



Gambar 3.1
Use Case Diagram Corgi Adventure

3.3.2 Use Case Skenario

Skenario adalah gambaran alur perjalanan yang menceritakan awal suatu aksi aktor dan program bereaksi sampai pada akhir program tersebut berakhir atau selesai melakukan fungsinya.

1. Use Case : *New Game*

   Aktor : Player

Pre-condition : *player* berada pada scene *main menu* dan memilih tombol *new game*.

Post-condition : *game* membuat *data* baru dan menampilkan scene *tutorial game*.

Definisi : untuk memulai permainan dengan *data* baru.

Tabel 3.1
Spesifikasi *Use Case* : *New Game*

Aksi Aktor	Reaksi Sistem
1. Player menekan tombol *new game*.	
	2. Sistem melakukan pengecekan apakah terdapat *save game*.
	3. Sistem memlakukan pembuatan data *player* baru
	4. Sistem menampilan stage *tutorial* game.

Alternatif 1 : terdapat *saved game* dan *save game overwrite* dengan *data* yang baru.

Tabel 3.2
Spesifikasi Alternatif 1 *Use Case* : *New Game*

Aksi Aktor	Reaksi Sistem
1 – 2 Sama dengan skenario normal.	
	3. Sistem akan menampilkan panel *overwrite*
4. *Player* menekan tombol YES.	
	5. Sistem melakukan pembuatan *data player* baru.
	6. Sistem menampilkan stage *tutorial game*.

Alternatif 2 : terdapat *saved game* dan save game tidak dihapus atau *overwrite* dengan *data* yang baru.

Tabel 3.3
Spesifikasi Alternatif 2 *Use Case* : *New Game*

Aksi Aktor	Reaksi Sistem
1 – 2 Sama dengan skenario normal.	
	3. Sistem melakukan pengecekan apakah terdapat *save game*.
	4. Sistem akan menampilkan *panel overwrite*
5. *Player* menekan tombol NO.	
	6. Panel *overwrite* ditutup.

2. Use Case : *Load Game*

Aktor : *Player*

Pre-condition : *player* berada pada *scene main menu* dan memilih tombol *load game*.

Post-condition : data berhasil di-*load* dan *scene level* select ditampilkan.

Definisi : untuk melanjutkan permainan dengan data yang sudah disimpan.

Tabel 3.4
Spesifikasi Use Case : *Load Game*

Aksi Aktor	Reaksi Sistem
1. Player menekan tombol *load game*.	
	2. Sistem melakukan *load data player*.
	3. Sistem menampilan *scene level select*.

3. Use Case : *Play Game*

Aktor : *Player*

Pre-condition : *player* berada pada *scene game menu* dan memilih tombol *stage* yang telah terbuka.

Post-condition : panel *result* ditampilkan dan hasil permainan tersimpan.

Definisi : untuk bermain *game*.

Tabel 3.5
Spesifikasi Use Case : *Play Game*

Aksi Aktor	Reaksi Sistem
1. *Player* menekan tombol *stage*.	
	2. Sistem melakukan *load data stage*, *level*, dan *record player*.
	3. *Scene stage game* ditampilkan dengan *spawn monster* dan *coin*.
4. *Player* menekan tombol *arrow* (*up*, *down*, *left*, *right*).	
	5. Karakter bergerak kiri dan kekanan; Karakter melihat keatas dan kebawah.
6. *Player* menekan tombol *jump*.	
	7. Karakter melompat sesuai arah posisi.
8. *Player* menekan tombol *fire*.	
	9. Karakter menembakan senjata.
10. *Player* menekan tombol *run*.	
	11. Karakter berlari sesuai arah posisi.
12. *Player* menekan tombol *jetpack*.	
	13. Karakter menggunakan jetpack sesuai arah.

Alternatif 1 : *Player* menekan tombol *pause* dan menekan tombol *continue*.

Tabel 3.6
Spesifikasi Alternatif 1 Use Case : Play Game

Aksi Aktor	Reaksi Sistem
1 – 13 Sama dengan skenario normal.	
14. *Player* menekan tombol *pause*.	
	15. *Game di-pause* dan *panel pause* ditampilkan.
16. *Player* menekan tombol *continue*.	
	17. *Game* dijalankan kembali.

Alternatif 2 : *Player* menekan tombol *pause* dan menekan tombol *restart*.

Tabel 3.7
Spesifikasi Alternatif 2 *Use Case : Play Game*

Aksi Aktor	Reaksi Sistem
1 – 13 Sama dengan skenario *normal*.	
14. *Player* menekan tombol *continue*.	
	15. *Game di-pause* dan *panel pause* ditampilkan.
16. *Player* menekan tombol *restart*.	
	17. *Stage Level* diulang dari awal.

Alternatif 3 : *Player* menekan tombol *pause* dan menekan tombol *save* & *main*.

Tabel 3.8
Spesifikasi Alternatif 3 *Use Case : Play Game*

Aksi Aktor	Reaksi Sistem
1 – 11 Sama dengan skenario normal.	
12. *Player* menekan tombol *pause*.	
	13. *Game* di-*pause* dan *panel pause* ditampilkan.
14. *Player* menekan tombol *save* & *menu*.	
	15. *Scene level* ditutup dan *scene level select* ditampilkan.

4. Use Case : *About*

   Aktor : *Player*

   Pre-condition : *player* berada pada *main menu* dan memilih tombol *about*.

   Post-condition : panel about ditutup.

   Definisi : untuk menampilkan informasi tentang *game*.

Tabel 3.9
Spesifikasi Use Case : About

Aksi Aktor	Reaksi Sistem
1. *Player* menekan tombol *about*.	
	2. Sistem menampilkan panel *about*.
3. *Player* menekan tombol *close*.	
	4. Sistem menutup panel *about*.

3.3.3 Activity Diagram

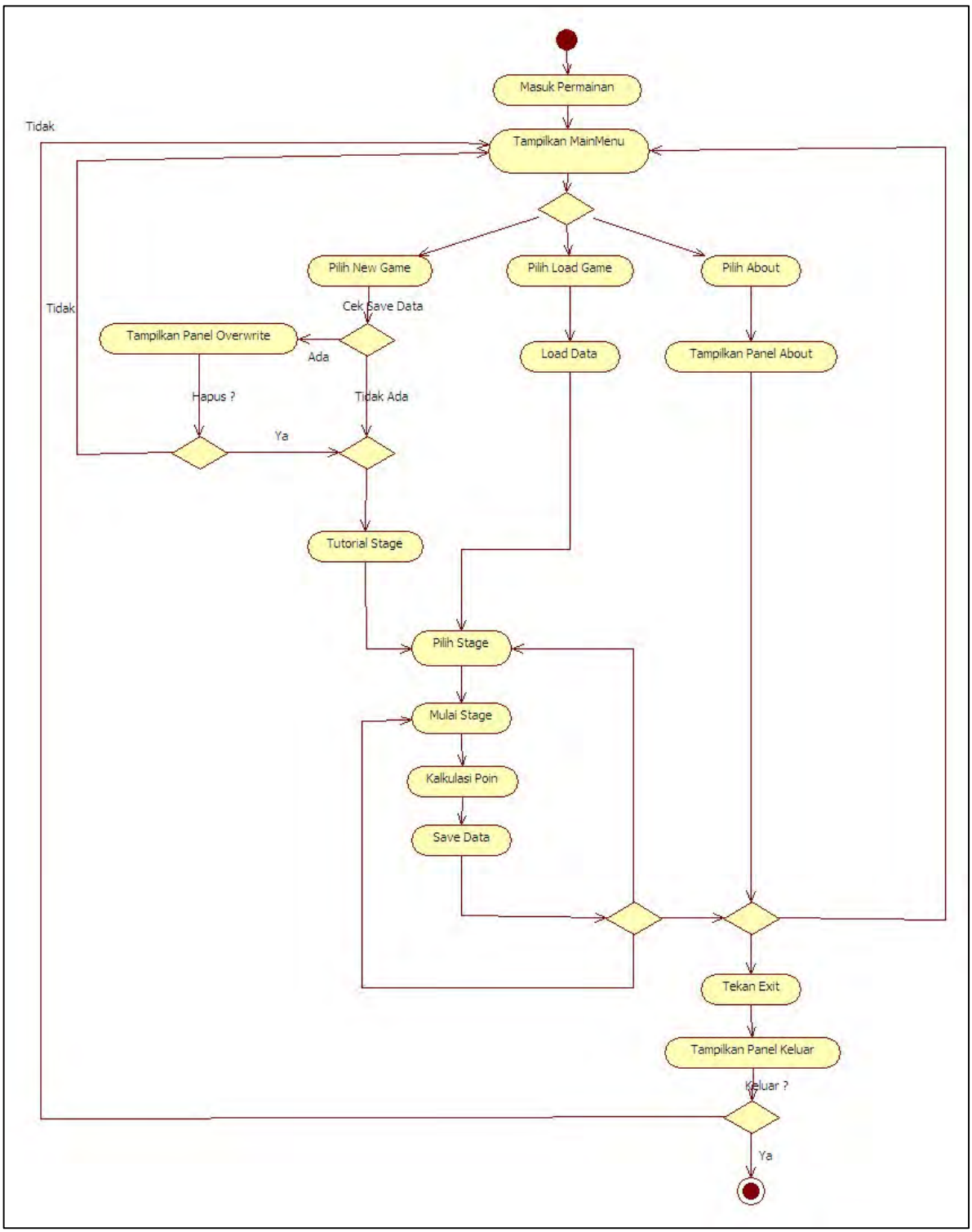
Activity diagram ini menjelaskan dari awal *game* dijalankan hingga *game* berakhir. Pertama kali dijalankan Sistem akan menampilkan Tampilan *Main Menu*, Apabila *Player* pertama kali menjalankan game ini maka akan tampil 2 *Menu* utama dan apabila *Player* pernah memainkan *game* ini sebelumnya makan akan tampil 3 *Menu* Utama.

Pada *Main Menu* Jika *Player* memilih *New Game*, pertama kali sistem akan mengecek apakah ada *data* permainan jika ada maka akan menampilkan panel *overwrite*. Selanjutnya *player* dapat memilih yes, maka *Player* akan dipindahkan ke *Tutorial Stage* dan apabila *player* memilih *no*, maka *player* akan kembali ke *Main Menu*.

Jika *Player* memilih *Load Game*, pertama kali sistem akan meload *data* yang ada pada *playerperf* dan setelah itu *player* akan dipindahkan ke *Level Select* dengan data yang telah *diload*. Pada *Level Select*, *player* dapat memilih *stage* yang sebelumnya pernah dimainkan atau meneruskan *stage* yang belum pernah dimainkan.

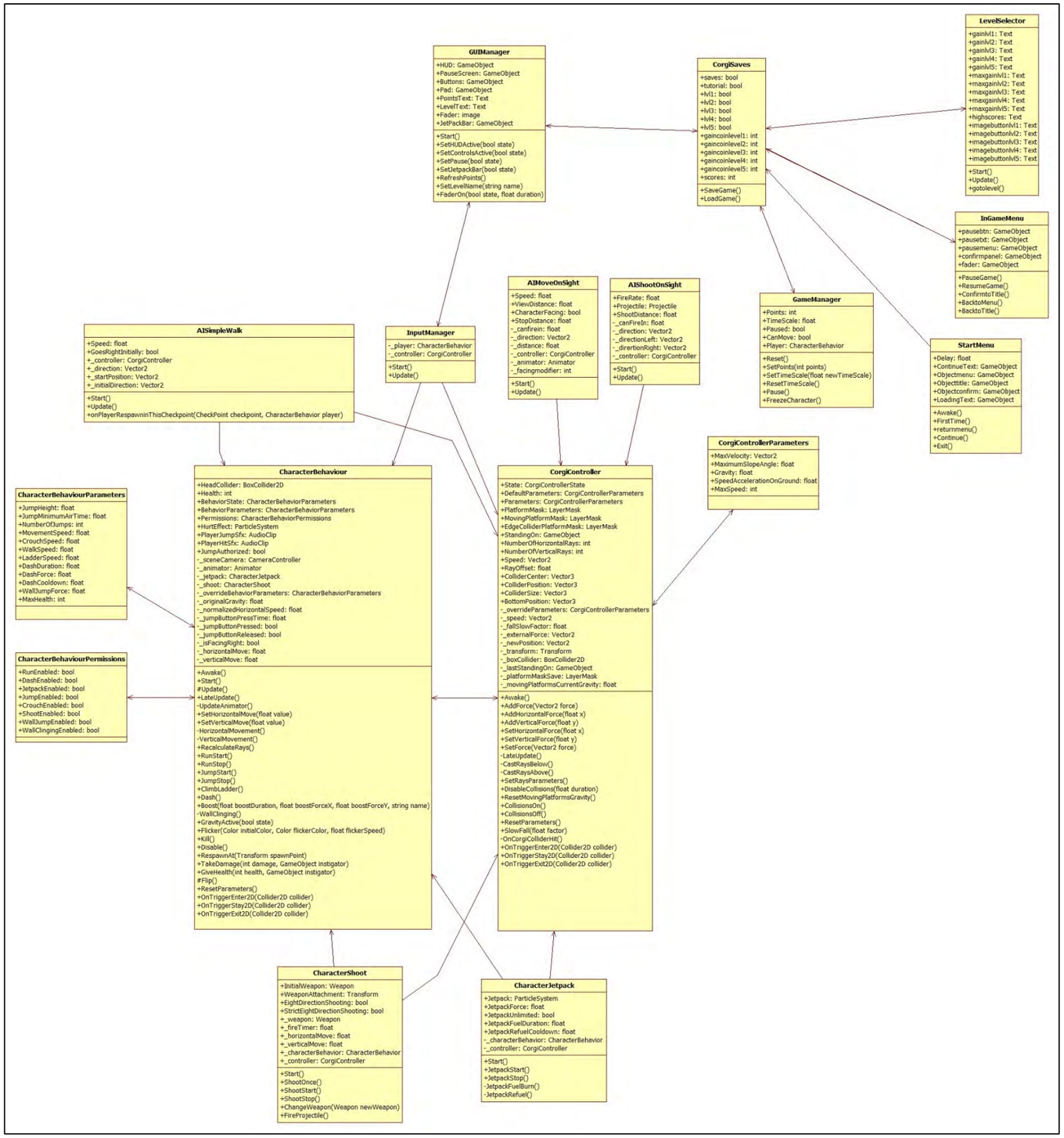
Jika *Player* memilih *About*, maka sistem akan menampilkan panel *About*. Panel *about* menampilkan informasi tentang *Game* mulai dari versi hingga pada pembuat *developer* yang membuat *game* tersebut.

Pada saat *Main Game*, sistem akan menyediakan kemudahan permainan sesuai dengan *level*. Player hanya dapat meneruskan *level* apabila *level* sebelumnya telah berhasil dimainkan dengan mencari bendera kuning pada akhir *stage*. Selama *stage* atau level berlangsung sistem akan mengkalkulasi point yang didapat dari mendapatkan *coin* dan membunuh musuh. Setelah *player* mencapai akhir *stage*, sistem akan secara otomatis akan menyimpan *data* yang telah didapatkan oleh *player*. Apabila *player* memilih *save & menu* maka sistem akan menyimpan *data player* dan mengembalikan *player* ke *Level Select*. Seperti Gambar 3.2.



Gambar 3.2
Activity Diagram CorgiAdventure

3.3.4 Class Diagram



Gambar 3.3
Class Diagram Corgi Adventure

Pada class diagram di Gambar 3.3 terdiri dari 17 class yang diantaranya terdapat 5 *main class* yaitu *CharacterBehaviour, CharacterBehaviourPermissions, CharacterBehaviourParameters, CorgiController, CorgiControllerParameters*, yang mendefinisikan *player behaviour* atau kegiatan yang dapat dikerjakan oleh player.

CharacterBehaviour, CharacterBehaviourPermissions, dan *CharacterBehaviourParameters* saling berkaitan karena *CharacterBehaviourPermission* menyimpan *permission* yang dapat dikerjakan oleh *CharacterBehaviour* seperti player dapat berlari atau tidak maupun player dapat menggunakan jetpack atau tidak dan *CharacterBehaviourParameter* menyimpan karakteristik dari *CharacterBehaviourPermission* seperti seberapa banyak player dapat melompat hingga seberapa tinggi player dapat melompat.

CorgiController merupakan class yang mengendalikan object berkaitan dengan collision antara moving object seperti player atau musuh dengan platform pada game. CorgiControllerParameters menyimpan parameter yang ada pada CorgiController seperti gravitasi seorang player maupun *angle* dalam bentuk derajat yang membuat karakter dapat melewati sebuah platform yang telah ditentukan.

AISimpleWalk, *AIMoveOnSight*, dan *AIShootOnSight* merupakan *class* yang mengimplementasikan AI dimana class ini yang berperan menggerakan musuh dengan menggunakan algoritma FSM.

CharacterShoot dan CharacterJetpack merupakan class yang memberikan sebuah karakter senjata ataupun jetpack yang keduanya saling berkaitan menggunakan CorgiController dan CharacterBehaviour.

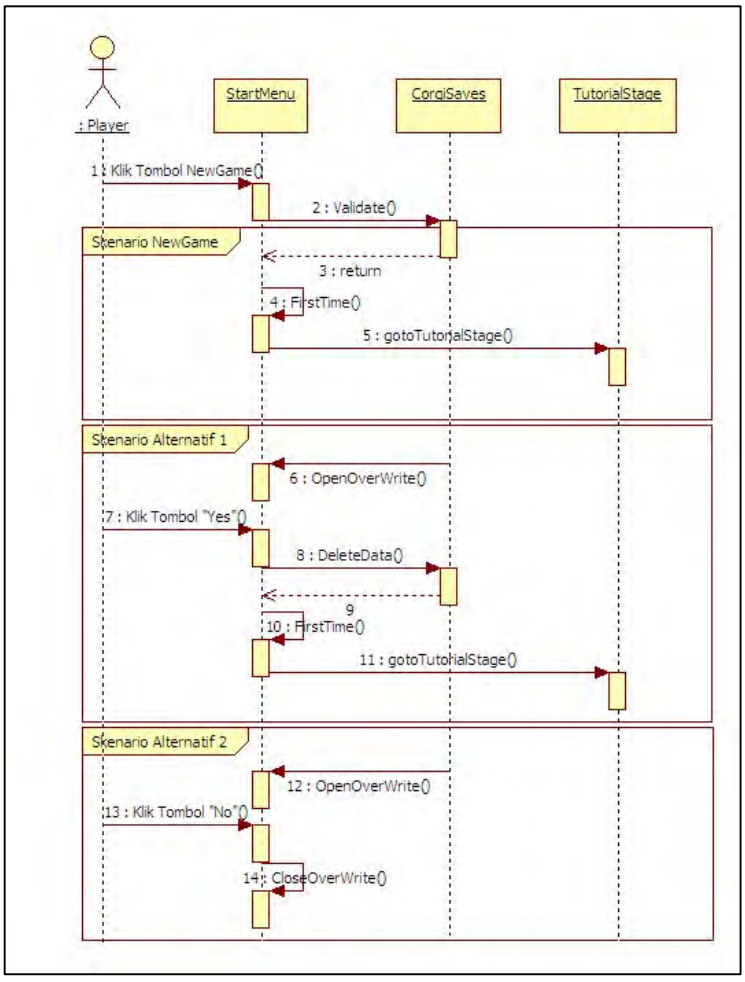
InputManager, *GUIManager*, *GameManager* merupakan *class* yang saling berkaitan dalam tampilan. *Class InputManger* sebuah *class* yang menagani *input* dari *player* seperti *input arrow button* maupun *input action button* seperti *jump button*, *dash button*, *fire button* dan *jetpack button*. *GUIManager* merupakan *class* yang menangani tampilan GUI seperti tampilan *HealthBar*, *score*, dan *JetpackEnergyBar*. *GameManager* merupakan *class* yang menangani waktu dan point, seperti waktu yang sedang berjalan

dapat dihentikan sewaktu-waktu dan pengkalkulasian point yang didapat dalam satu stage atau level.

StartMenu, *LevelSelector*, dan *InGameMenu* merupakan class yang berhubungan dengan scene menu yang sedang berjalan seperti fungsi button pada startmenu. Class CorgiSave menyimpan semua yang berhubungan dengan point, unlocked level hingga seberapa banyak coin yang player dapatkan ketika menempuh salah satu stage tersebut.

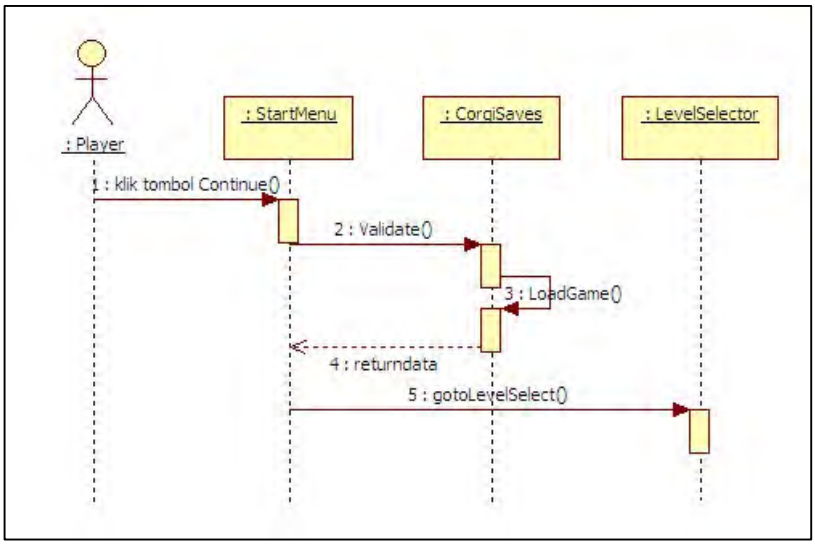
3.3.5 Sequence Diagram

Pada saat *Player* menekan Tombol *NewGame*, maka sistem akan mengecek apakah terdapat *save game* apabila Sistem menemukan *Save* yang ada pada *game* maka Panel *OverWrite* akan ditampilkan. Disini *player* dapat memilih untuk menimpa *save game* sebelumnya dengan yang baru atau tidak, jika *player* memilih ya maka sistem akan membuat *data* baru dan player akan dipindahkan ke *TutorialStage*, jika player tidak maka sistem akan menutup panel *OverWrite* dan kembali ke *MainMenu*. Seperti di gambar 3.4.



Gambar 3.4
Sequence Diagram New Game

Player yang telah bermain *game* ini sebelumnya dapat melanjutkan *game* dengan menggunakan *data* yang pernah dibuat. *Player* dapat memilih tombol *Continue*, maka sistem akan membaca *save game* dan ditampilkan di *Level selector* setelah itu *player* dipindahkan ke *Level Select*. Seperti Gambar 3.5.



Gambar 3.5
Sequence Diagram LoadGame

Player memilih stage level 1, sistem akan meng-inisialisasi yang ada GUI*Manager* dengan menampilkan GUI dan menampilkan *Level*. Pada saat *player* menekan *arrow button* sistem pertama kali akan mendeteksi *arrow button* yang ditekan apabila tombol *horizontal* maka sistem akan memanggil pergerakan secara horizontal, jika tombol *vertical* ditekan maka sistem akan memanggil pergerakan secara *vertical*. Apabila karakter dalam keadaan diam dan berada diatas *platform* maka sistem akan menggeser kamera kebawah lalu apabila dilepas maka kamera kembali ke posisi karakter begitu pula sebaliknya.

Ketika *Player* menekan *Action Button* sistem akan mendeteksi apakah yang ditekan oleh player. Tombol ini terdiri dari *Jump, Dash, Jetpack, Run*, dan *Fire*.

Jika tombol *Jump* ditekan maka sistem akan mendeteksi apakah *jump* dapat dilakukan atau tidak, apabila jump dapat dilakukan maka sistem akan melihat seberapa

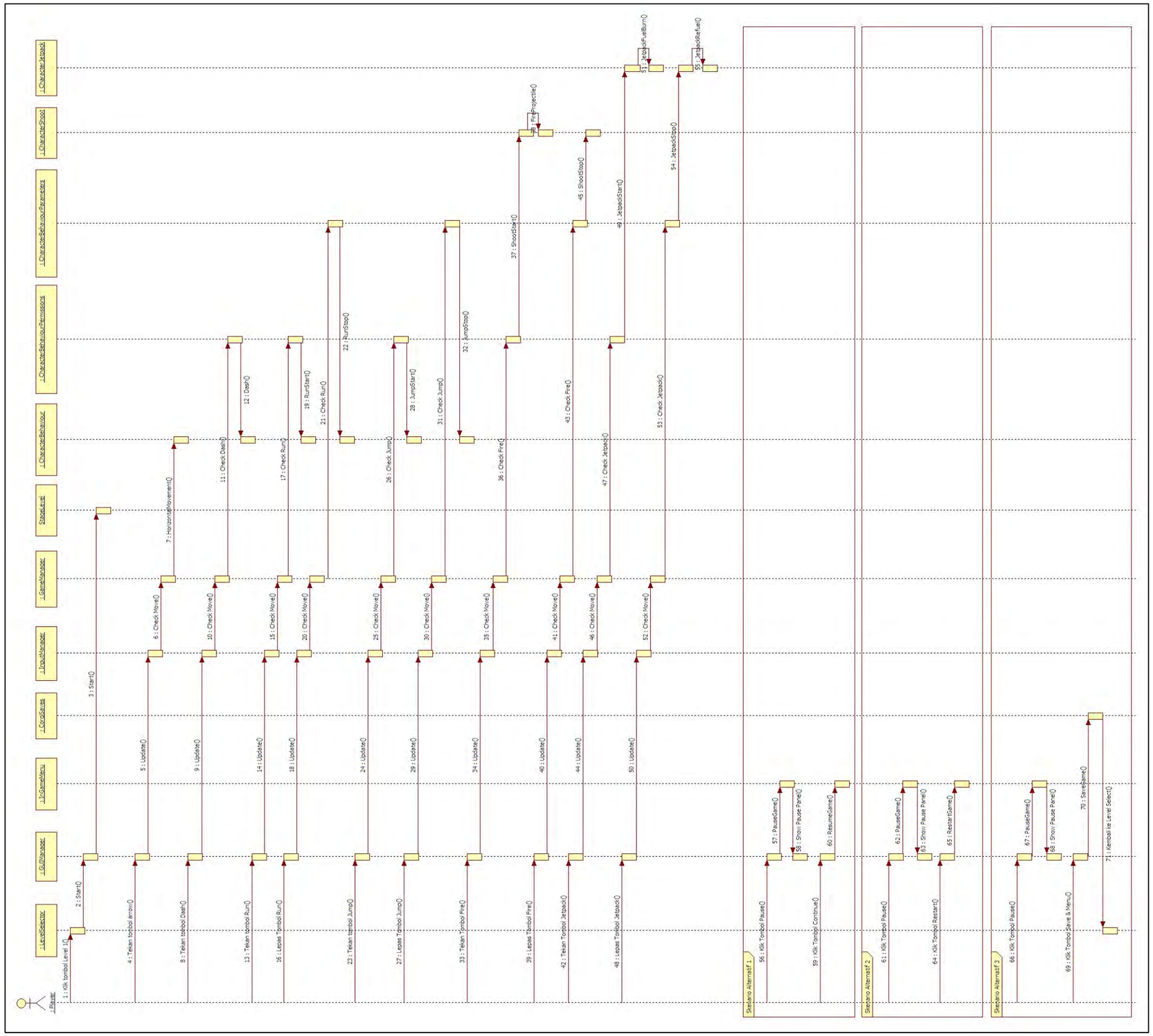
banyak karakter dapat melompat dan seberapa jauh karakter melompat dan apabila tidak dapat maka sistem tidak akan memberikan respon untuk fungsi tombol *jump*.

Jika tombol *Dash* ditekan maka sistem akan mendeteksi apakah *dash* dapat dilakukan atau tidak, apabila dash dapat dilakukan maka sistem akan melihat seberapa jauh karakter dapat melakukan *dash* dan apabila tidak dapat maka sistem tidak akan memberikan respon untuk fungsi tombol *dash*.

Jika tombol *Run* ditekan maka sistem akan mendeteksi apakah *run* dapat dilakukan atau tidak, apabila run dapat dilakukan maka sistem akan melihat seberapa lama tombol tersebut dengan pergerakan sesuau dengan lama tombol ditekan dan apabila tombol dilepas dapat maka sistem akan menghentikan karakter yang berlari.

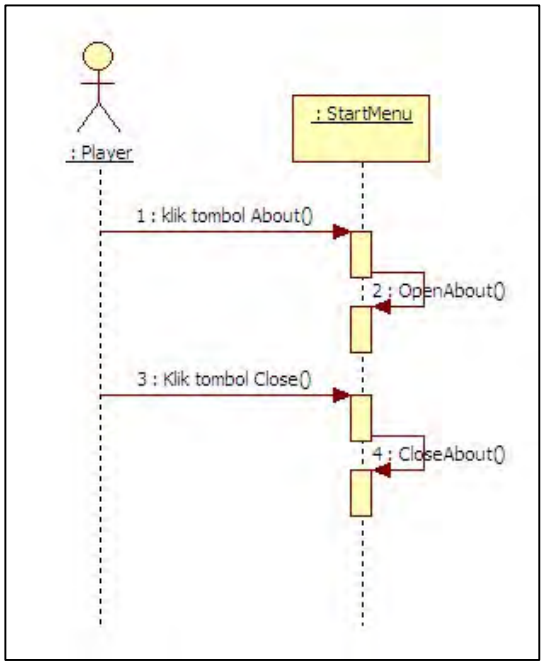
Jika tombol *Jetpack* ditekan maka sistem akan mendeteksi apakah *jetpack* dapat dilakukan atau tidak, apabila *jetpack* dapat dilakukan maka sistem akan melihat seberapa lama tombol tersebut ditekan dengan pergerakan karakter yang menggunakan jetpack dengan energi terbatas dan apabila tombol dilepas ataupun *energy* telah habis dapat maka sistem akan menghentikan karakter.

Jika tombol *Fire* ditekan maka sistem akan mendeteksi apakah menembak dapat dilakukan atau tidak, apabila dapat dilakukan maka sistem akan melihat seberapa lama tombol tersebut ditekan dengan menggerakan *projectile* peluru keluar dari senjata karakter dan apabila tombol dilepas dapat maka sistem akan menghentikan karakter. Seperti Gambar 3.6.



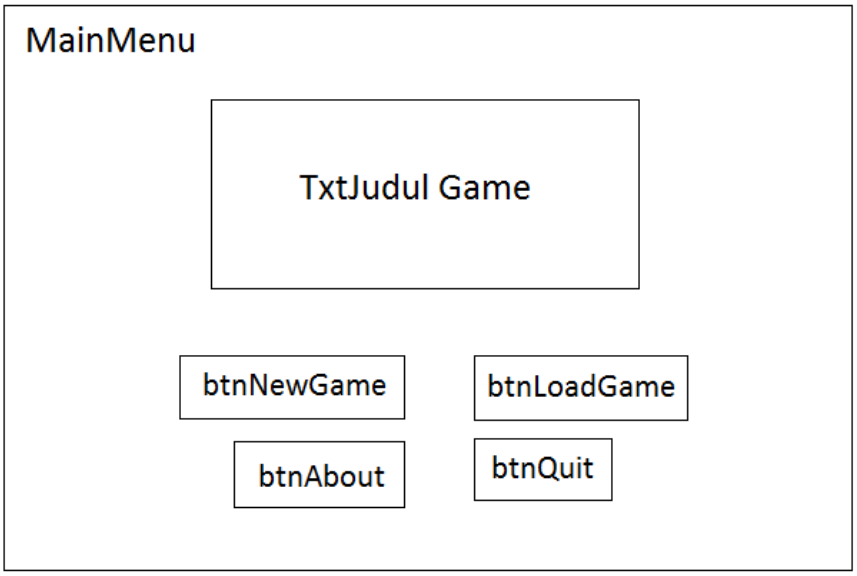
Gambar 3.6
Sequence Diagram Play Game

Player menekan tombol *about*, maka *about* panel akan muncul dan memperlihatkan informasi yang ada pada game, diantaranya versi dari *game* dan nama pembuat *game*. Seperti pada gambar 3.7.



Gambar 3.7
Sequence Diagram About

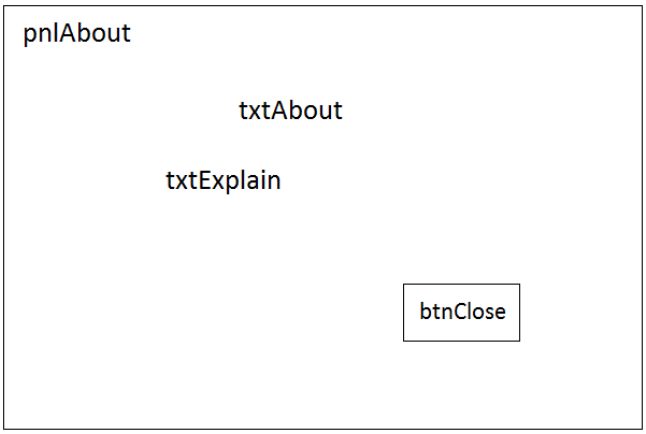
3.3.6 Rancangan Antar Muka



Gambar 3.8
Rancangan *MainMenu*

Penjelasan mengenai komponen – komponen pada Gambar 3.8 sebagai berikut :

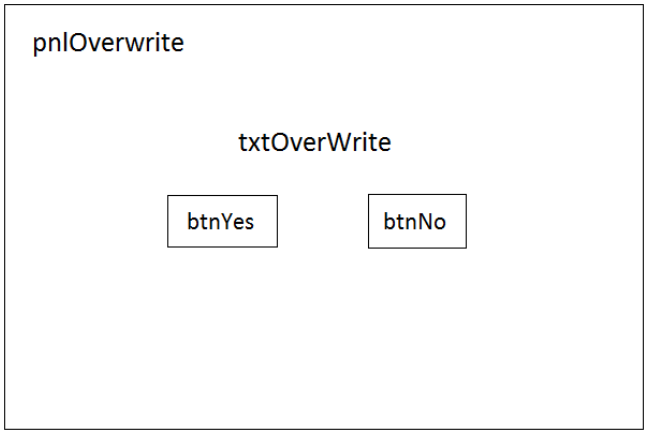
1. TxtJudulGame : berfungsi untuk menampilkan tulisan dari judul *game Corgi Adventure*.
2. btnNewGame : berfungsi untuk memulai *game* baru.
3. btnLoadGame : berfungsi untuk melanjutkan *game* yang sudah tersimpan sebelumnya.
4. btnAbout : berfungsi untuk menampilkan panel *about*.
5. btnQuit : berfungsi untuk keluar dari *game*.



Gambar 3.9
Rancangan Panel *About*

Penjelasan mengenai komponen – komponen pada Gambar 3.9 sebagai berikut :

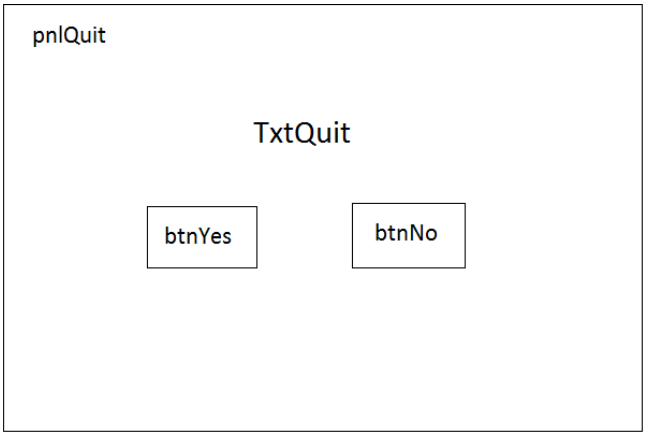
1. txtAbout : berfungsi untuk menampilkan tulisan tipe panel yang sedang terbuka.
2. btnClose : berfungsi untuk menutup panel *about*.
3. txtExplain : berfungsi untuk menampilkan tulisan penjelasan mengenai game *Corgi Adventure*



Gambar 3.10
Rancangan Panel Overwrite

Penjelasan mengenai komponen – komponen pada Gambar 3.10 sebagai berikut :

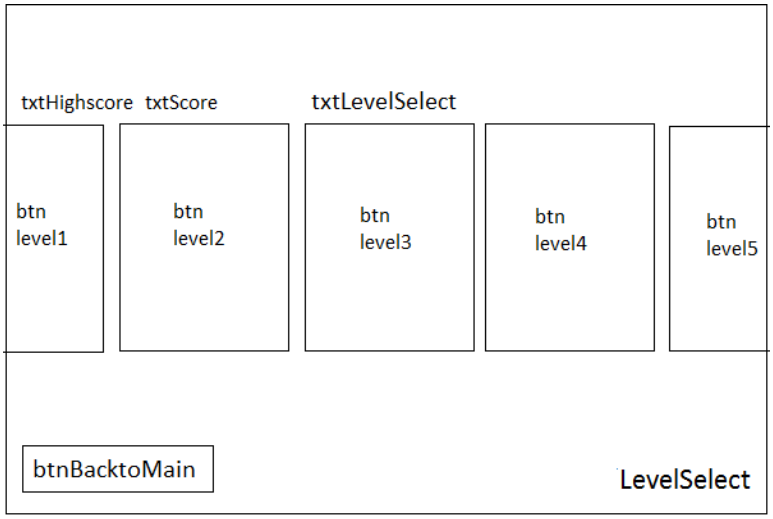
1. txtOverWrite : berfungsi untuk menampilkan tulisan untuk menimpa data game.
2. btnNo : berfungsi untuk menolak penghapusan data dan kembali ke mainmenu.
3. btnYes : berfungsi untuk menyetujui penghapusan data.



Gambar 3.11
Rancangan Panel Quit

Penjelasan mengenai komponen – komponen pada Gambar 3.11 sebagai berikut :

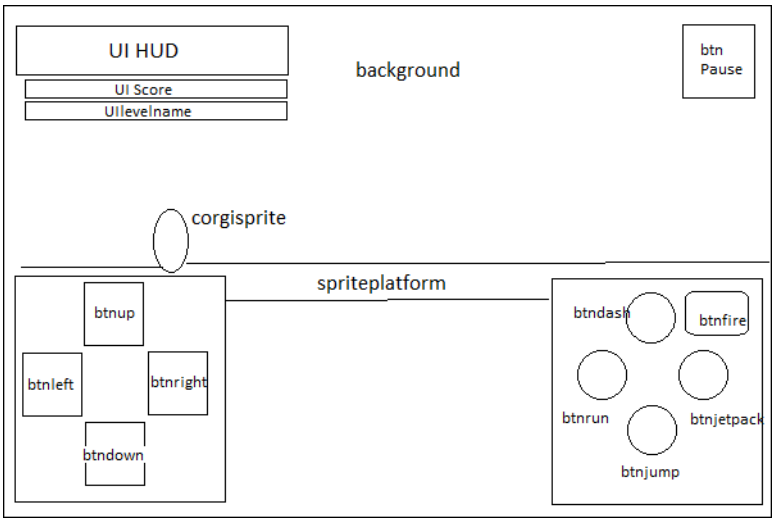
1. txtQuit : berfungsi untuk menampilkan tulisan tipe panel yang sedang terbuka.
2. btnNo : berfungsi untuk menolak keluar dari game dan kembali ke mainmenu.
3. btnYes : berfungsi untuk menyetujui keluar dari game.



Gambar 3.12
Rancangan Level Select

Penjelasan mengenai komponen – komponen pada Gambar 3.12 sebagai berikut :

1. txtHighscore : berfungsi untuk menampilkan tulisan highscore.
2. txtScore: berfungsi untuk menampilkan score yang sudah didapatkan oleh player.
3. txtLevelSelect : berfungsi untuk menampilkan tulisan Level Select.
4. btnlevel1 : berfungsi untuk mengganti level menuju level 1
5. btnlevel2 : berfungsi untuk mengganti level menuju level 2
6. btnlevel3 : berfungsi untuk mengganti level menuju level 3
7. btnlevel4 : berfungsi untuk mengganti level menuju level 4
8. btnlevel5 : berfungsi untuk mengganti level menuju level 5
9. btnbacktomain : berfungsi untuk kembali ke mainmenu

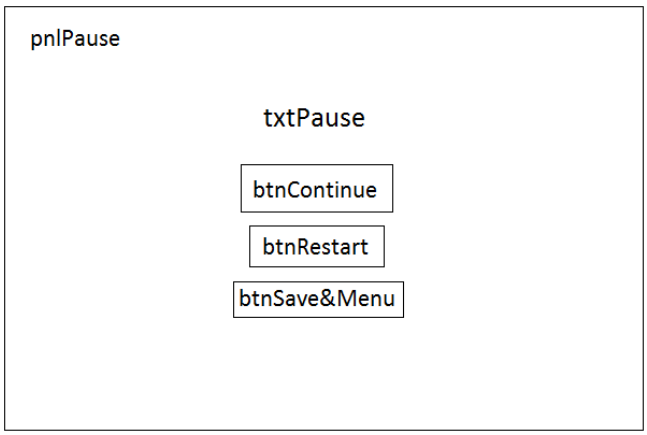


Gambar 3.13
Rancangan Interface pada main game

Penjelasan mengenai komponen – komponen pada Gambar 3.13 sebagai berikut :

1. UIHud : berfungsi untuk menampilkan health indicator dan jetpack energy.
2. UIScore: berfungsi untuk menampilkan score yang sudah didapatkan oleh player.
3. UIlevelname : berfungsi untuk menampilkan nama level yang sedang ditempuh.
4. btnPause : berfungsi untuk memberhentikan sementara permainan dan membuka panel pause.

5. btnup : berfungsi untuk mengarahkan character keatas.
6. btndown : berfungsi untuk mengarahkan character kebawah.
7. btnleft : berfungsi untuk mengarahkan character kekiri.
8. btnright : berfungsi untuk mengarahkan character kekanan.
9. btnjump : berfungsi untuk character yang dijalankan dapat melompat.
10. btnrun : berfungsi untuk character yang dijalankan dapat berlari.
11. btndash : berfungsi untuk character yang dijalankan dapat berlari secara tergesa gesa.
12. btnfire : berfungsi untuk character yang dijalankan dapat menembakan senjatanya.
13. btnfire : berfungsi untuk character yang dijalankan dapat menggunakan jetpack.
14. Background : berfungsi untuk menampilkan background pada game.
15. corgisprite : berfungsi untuk menampilkan sprite player pada game.
16. spriteplatform : berfungsi untuk menampilkan sprite platform pada game.



Gambar 3.14
Rancangan Panel pause

Penjelasan mengenai komponen – komponen pada Gambar 3.14 sebagai berikut :

1. txtPause : berfungsi untuk menampilkan tulisan pause.
2. btnContinue : berfungsi untuk kembali ke game.
3. btnRestart : berfungsi untuk mengulang level.
4. btnSave&Menu : berfungsi untuk save level dan keluar dari level.

BAB IV

IMPLEMENTASI DAN PENGUJIAN PERANGKAT LUNAK

4.1 Implementasi

Pada bab ini akan dijelaskan bagaimana seluruh perancangan dan analisa diimplementasikan untuk menghasilkan perangkat lunak game untuk platform Android.

4.1.1 Lingkungan Pengembangan

Lingkungan pengembangan adalah spesifikasi perangkat keras yang digunakan selama permainan ini dikembangkan. Pengembangan dilakukan di sebuah PC dengan spesifikasi sebagai berikut:

1. Processor : Intel Processor Core 2 Duo 3.6 GHz
2. RAM : DDR3 4GB
3. Layar : 21" dengan resolusi 1600x900
4. VGA : Nvidia Geforce GT 9400
5. HDD : 1 TB

Selain perangkat keras yang telah disebutkan, digunakan pula perangkat lunak yang mendukung dalam pengembangan game ini. Perangkat lunak tersebut yaitu :

1. Windows 8.1

   Sistem operasi yang digunakan adalah Windows 8.1. Sistem operasi ini di gunakan karena umum dipakai dan mendukung hamper semua software baru yang ada saat pengerjaan perangkat lunak yang dikembangkan pada Tugas Akhir ini.

2. Unity 5.1

   Unity 5.1 merupakan versi terbaru dari Unity game engine. Unity 5.1 memiliki kelebihan dalam pembuatan GUI dibanding dengan versi sebelumnya. Fitur–fitur pada Unity 5.1 hampir semuanya dapat digunakan tanpa perlu membeli lisensi profesional.

3. DirectX 11

   Aplikasi framework yang berfungsi untuk menangani berbagai tugas dalam permrograman multimedia khususnya game.

4. Android SDK

   Android SDK dipakai dalam pengembangan perangkat lunak ini karena Android SDK merupakan tools utama yang diperlukan dalam pembuatan perangkat lunak untuk Android.

5. JDK 8.0

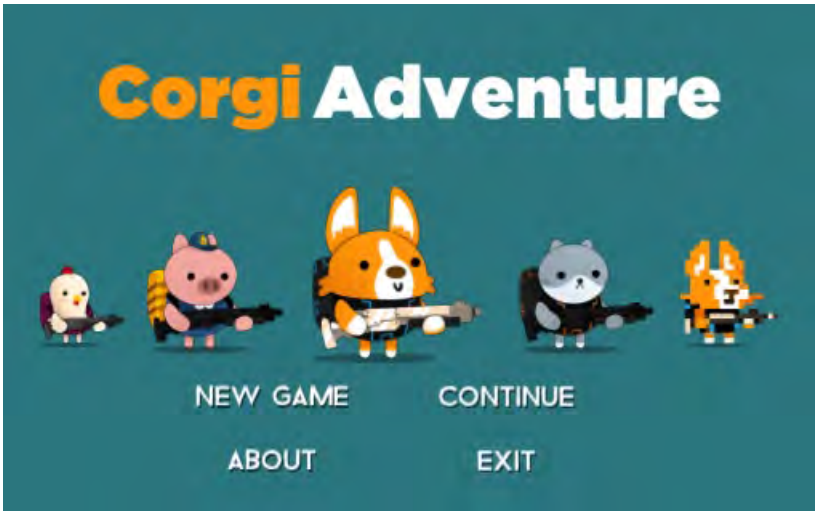
   JDK 8.0 digunakan juga sebagai tools tambahan selain Android SDK dalam pembuatan perangkat lunak menjadi extensi .apk.

4.1.2 Tampilan Antar Muka

Berikut adalah hasil screenshot dari seluruh kemungkinan interface yang ada di dalam game ini :

1. Main menu

   Saat pertama kali di jalankan, game akan menampilkan tampilan main menu seperti pada Gambar 4.1.



Gambar 4.1
Tampilan MainMenu

Pada main menu terdapat empat buah tombol pada tampilan. Tombol-tombol itu adalah :

a. Tombol New Game

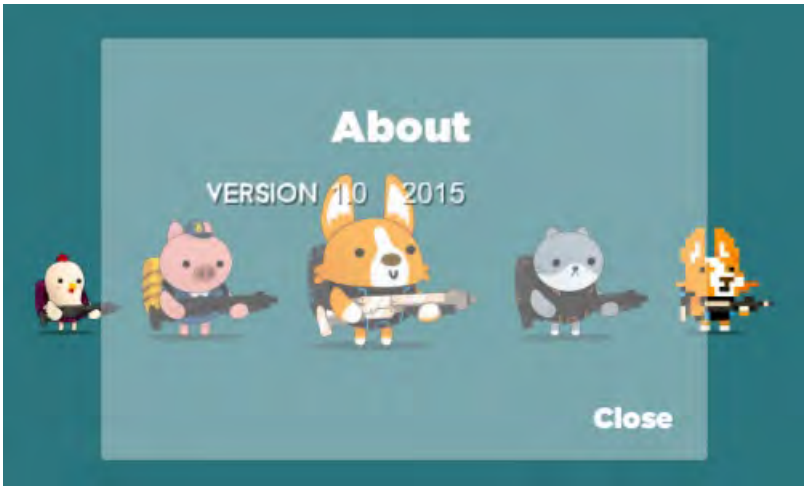
Tombol ini berfungsi untuk membuat permainan baru. Namun jika ternyata ada *saves game* sebelumnya yang tersimpan, maka akan secara otomatis menampilkan panel overwrite seperti pada Gambar 4.2.

Gambar 4.2
Tampilan Overwrite

b. Tombol Continue

Tombol ini berfungsi untuk melanjutkan permainan yang sudah tersimpan sebelumnya.

c. Tombol About

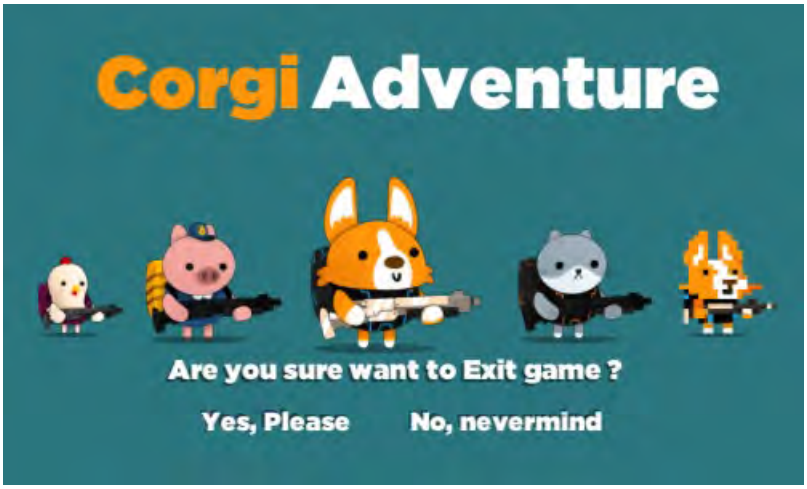


Gambar 4.3
Tampilan About

Tombol ini berfungsi menampilkan panel about seperti pada Gambar 4.3. Panel about berisi informasi tentang permainan ataupun versi game yang sedang dijalankan.

d. Tombol Quit

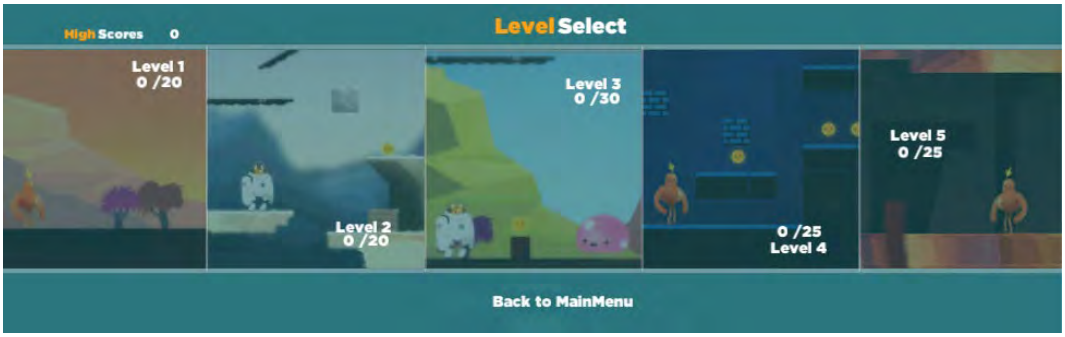
Tombol ini berfungsi untuk menutup/keluar dari game dan menampilkan panel quit seperti pada Gambar 4.4.



Gambar 4.4
Tampilan Quit Game

2. Level Select

Level Select merupakan menu pemilihan level yang akan terbuka setelah tombol continue ditekan dari main menu seperti pada Gambar 4.5.



Gambar 4.5
Tempilan Level Select

Pada menu ini terdapat beberapa tombol, yaitu :

a. Tombol Level 1

Tombol ini berfungsi untuk berpindah ke level 1.

b. Tombol Level 2

Tombol ini berfungsi untuk berpindah ke level 2.

c. Tombol Level 3

Tombol ini berfungsi untuk berpindah ke level 3.

d. Tombol Level 4

Tombol ini berfungsi untuk berpindah ke level 4.

e. Tombol Level 5

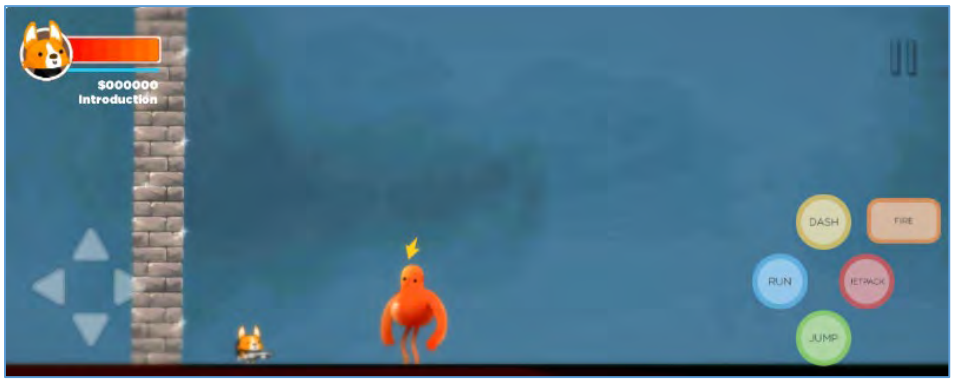
Tombol ini berfungsi untuk berpindah ke level 5.

f. Tombol Back to Menu

Tombol ini berfungsi untuk kembali ke main menu.

3. Main game

Main game dapat diakses saat player memilih stage yang bisa diakses saat berada pada level select, apabila player pertama kali memainkan game ini maka player akan berpindah secara otomatis ke level tutorial dan apabila player telah memainkan game ini ingin melanjutkan kembali game sebelumnya maka akan berpindahke level select. Tampilan main game seperti Gambar 4.6.



Gambar 4.6
Tampilan Main Game

Pada main game terdapat sebelas buah tombol yang memiliki fungsi masing-masing, yaitu :

a. Tombol pause

Tombol ini berfungsi untuk membuka panel pause seperti pada Gambar 4.7 dan menghentikan sementara permainan. Pada panel pause sendiri terdapat tiga tombol yaitu tombol continue unruk melanjutkan permainan dan menutup panel

pause, tombol restart untuk mengulang kembali stage tersebut, tombol save & menu untuk save game dan kembali ke level select.

Gambar 4.7
Tampilan Panel Pause

b. Tombol vertical arrow (left dan right arrow)

Tombol ini berfungsi untuk menggerakan karakter ke arah vertical yaitu ke arah kanan dan kearah kiri.

c. Tombol horizontal arrow (up dan down arrow)

Tombol ini berfungsi untuk menggerakan karakter ke arah horizontal yaitu keatas dan kebawah.

d. Tombol Jump

Tombol ini berfungsi untuk menggerakan karakter dengan cara melompat.

e. Tombol Run

Tombol ini berfungsi untuk menggerakan karakter dengan cara berlari panjang

f. Tombol Dash

Tombol ini berfungsi untuk menggerakan karakter dengan cara berlari pendek

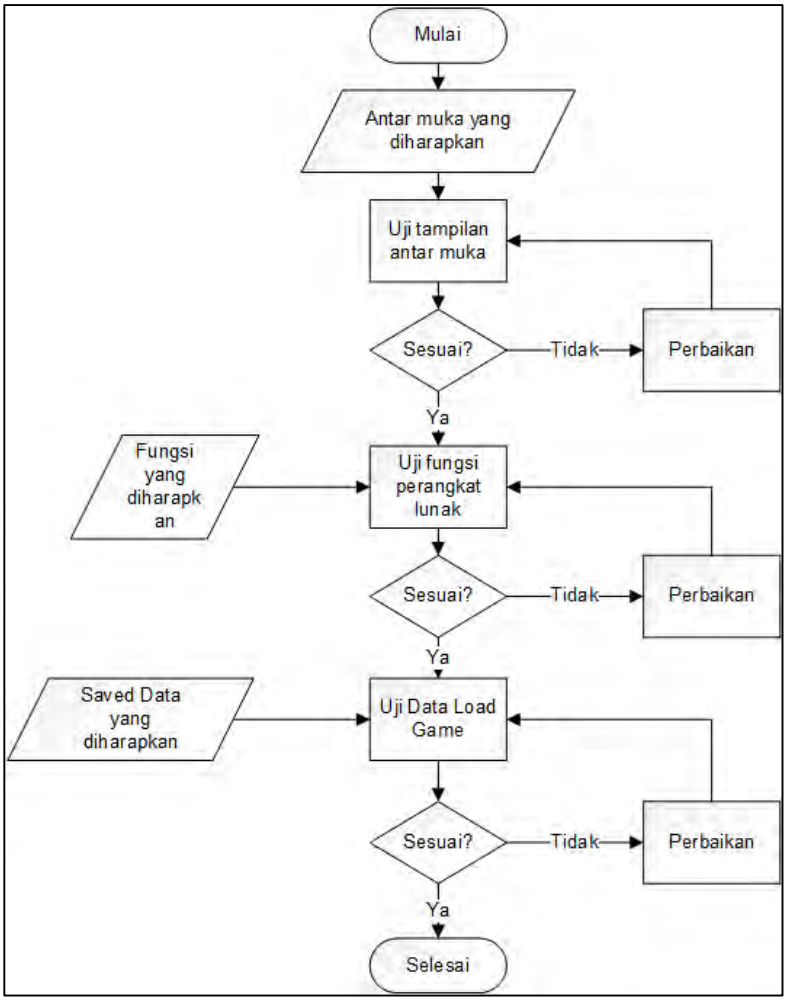
g. Tombol Jetpack

Tombol ini berfungsi untuk menggerakan karakter dengan menggunakan jetpack.

h. Tombol Fire

Tombol ini berfungsi untuk menembakan senjata yang ada pada karakter.

4.2 Pengujian



Gambar 4.8
Skema Pengujian

Pada Gambar 4.8, Pengujian yang dilakukan pada perangkat lunak ini adalah pengujian tampilan antar muka, pengujian fungsi perangkat lunak, dan pengujian data load game. Pertama, pengujian tampilan antar muka akan menguji ketepatan setiap antar muka yang ada pada sistem berikut komponen yang ada didalamnya. Pengujian fungsi perangkat lunak, menguji setiap fungsi yang ada pada perangkat lunak. Jika sebuah fungsi menghasilkan hasil sesuai dengan yang diharapkan, maka fungsi tersebut dinyatakan berhasil. Pada tahap pengujian data load game, menguji save data yang dihasilkan perangkat lunak. Jika saved data yang dihasilkan perangkat lunak berdasarkan data yang telah player dapatkan, maka pengujian data load game ini berhasil. Setelah tahap pengujian selesai, semua fungsi dan antar muka yang tidak berhasil akan dicari

kesalahannya dan diperbaiki, sedangkan pengujian data load game menentukan tingkat keberhasilan perangkat lunak ini.

4.2.1 Kondisi Pengujian

Pengujian menggunakan metode black box yang berfokus melakukan pengujian melalui tampilan perangkat lunak. Pengujian akan menguji fungsi – fungsi dari game, apakah sudah berjalan sebagaimana seharusnya atau tidak dengan melihat hasil output dari masing – masing percobaan uji. Fungsi – fungsi yang akan diuji dalam pengujian adalah sebagai berikut :

1. Navigasi main menu.
2. Navigasi Level Select.
3. Proses bermain pada main game.
4. Kompatibilitas pada platform Android.
5. Resolusi layar.

4.2.2 Pelaksanaan Pengujian

1. Navigasi main menu

Tabel 4.1
Tabel pengujian navigasi main menu

No	Jenis Pengujian	Ekspektasi	Output	Ket
1	Menu utama pada saat pertama kali dijalankan	Menu utama ditampilkan	Menu utama ditampilkan	BERHASIL
2	Tombol new game ditekan saat tidak ada data di player perf	Game menu ditampilkan	Game menu ditampilkan	BERHASIL
3	Tombol new game ditekan saat ada data di player perf	Panel overwrite ditampilkan	Panel overwrite ditampilkan	BERHASIL
4	Tombol yes pada panel overwrite ditekan	Data dihapus dan game menu ditampilkan	Data dihapus dan game menu ditampilkan	BERHASIL
5	Tombol no pada panel overwrite ditekan	Data tidak jadi dihapus dan panel overwrite ditutup	Data tidak jadi dihapus dan panel overwrite ditutup	BERHASIL
6	Tombol close pada overwrite panel ditekan	Panel overwrite ditutup	Panel overwrite ditutup	BERHASIL
7	Tombol load game ditekan	Load data dan Level Select ditampilkan	Load data dan Level Select ditampilkan	BERHASIL
8	Tombol about ditekan	Panel about ditampilkan	Panel about ditampilkan	BERHASIL
9	Tombol close pada panel about ditekan	Panel about ditutup	Panel about ditutup	BERHASIL

No	Jenis Pengujian	Ekspektasi	Output	Ket
10	Tombol quit ditekan	Panel quit ditampilkan	Panel quit ditampilkan	BERHASIL
11	Tombol yes pada panel quit ditekan	Keluar dari game	Keluar dari game	BERHASIL
12	Tombol no pada panel quit ditekan	Tidak jadi keluar dari game dan panel quit ditutup	Tidak jadi keluar dari game dan panel quit ditutup	BERHASIL
13	Tombol close pada panel quit ditekan	Panel quit ditutup	Panel quit ditutup	BERHASIL

2. Navigasi level select

Tabel 4.2
Tabel pengujian navigasi level select

No	Jenis Pengujian	Ekspektasi	Output	Ket
1	Player telah menyelesaikan tutorial stage atau load game ditekan dari main menu	Level Select ditampilkan	Level Select ditampilkan	BERHASIL
2	Tombol back to menu ditekan	Kembali ke main menu	Kembali ke main menu	BERHASIL
3	Tombol level 1 ditekan	Berpindah ke scene permainan level 1	Berpindah ke scene permainan level 1	BERHASIL
4	Tombol level 2 ditekan	Berpindah ke scene permainan level 2	Berpindah ke scene permainan level 2	BERHASIL
5	Tombol level 3 ditekan	Berpindah ke scene permainan level 3	Berpindah ke scene permainan level 3	BERHASIL
6	Tombol level 4 ditekan	Berpindah ke scene permainan level 4	Berpindah ke scene permainan level 4	BERHASIL
7	Tombol level 5 ditekan	Berpindah ke scene permainan level 5	Berpindah ke scene permainan level 5	BERHASIL

3. Proses bermain pada main game

Tabel 4.3
Tabel Pengujian bermain pada main game

No	Jenis Pengujian	Ekspektasi	Output	Ket
1	Main game saat tombol level ditekan dari Level Select	Load data score yang sebelumnya pernah didapatkan	Load data score dari yang pernah dimainkan	BERHASIL
2	Tombol pause ditekan	Panel pause ditampilkan dan game dipause	Panel pause ditampilkan dan game dipause	BERHASIL
3	Tombol close pada panel pause ditekan	Panel pause ditutup dan game dilanjutkan	Panel pause ditutup dan game dilanjutkan	BERHASIL
4	Tombol resume pada panel pause ditekan	Panel pause ditutup dan game dilanjutkan	Panel pause ditutup dan game dilanjutkan	BERHASIL
5	Tombol restart pada panel pause ditekan	Panel pause ditutup dan game diulang dari awal	Panel pause ditutup dan game diulang dari awal	BERHASIL
6	Tombol exit level pada panel pause ditekan	Panel pause ditutup dan kembali ke main menu	Panel pause ditutup dan kembali ke main menu	BERHASIL

No	Jenis Pengujian	Ekspektasi	Output	Ket
7	Tombol kiri ditekan	Karakter bergerak ke kiri dan kamera mengikuti karakter	karakter bergerak ke kiri dan kamera mengikuti karakter	BERHASIL
8	Tombol kanan ditekan	Karakter bergerak ke kanan dan kamera mengikuti karakter	karakter bergerak ke kanan dan kamera mengikuti karakter	BERHASIL
9	Tombol atas ditekan	Karakter melihat keatas dan kamera bergeser sedikit kebagian atas karakter	karakter melihat keatas dan kamera bergeser sedikit kebagian atas karakter	BERHASIL
10	Tombol bawah ditekan	Karakter melihat keatas dan kamera bergeser sedikit kebagian bawah karakter	karakter melihat keatas dan kamera bergeser sedikit kebagian bawah karakter	BERHASIL
11	Tombol Jump ditekan	Karakter akan melompat sesuai arah	Karakter melompat sesuai arah	BERHASIL
12	Tombol Dash ditekan	Karakter akan berlari dash sesuai arah	Karakter berlari dash sesuai arah	BERHASIL
13	Tombol Run ditekan	Karakter akan berlari sesuai dengan arah	Karakter berlari sesuai dengan arah	BERHASIL
14	Tombol Jetpack ditekan	Karakter akan terbang menggunakan jetpack	Karakter terbang menggunakan jetpack	BERHASIL
15	Tombol Fire ditekan	Karakter akan menembakan senjatanya	Karakter menembakan senjatanya	BERHASIL
16	Karakter mendapatkan coin	Karakter mendapatkan coin apabila telah bertabrakan dengan object coin dan point akan bertambah 10	Karakter bertabrakan dengan object coin dan point bertambah 10	BERHASIL
17	Karakter mendapatkan stimpack	Karakter mendapatkan stimpack apabila telah bertabrakan dengan object stimpack dan health bertambah 25 point	Karakter bertabrakan dengan object stimpack dan health bertambah 25 point	BERHASIL
18	Karakter mendapatkan weapon upgrade	Karakter mendapatkan weapon upgrade apabila telah bertabrakan dengan object weapon upgrade dan projectile senjata berubah menjadi laser	Karakter bertabrakan dengan object weapon upgrade dan projectile senjata berubah menjadi laser	BERHASIL
19	Karakter mendapatkan damage dari musuh	Karakter mendapatkan damage dari projectile atau bertabrakan dengan musuh dan UI Health bar berkurang	Karakter mendapatkan damage dari projectile atau bertabrakan dengan musuh dan UI Health bar berkurang	BERHASIL

4. Kompatibilitas pada platform Android

Tabel 4.4
Tabel pengujian kompatibilitas

No	Jenis Pengujian	Ekspektasi	Output	Ket
1	Game di-install pada perangkat Samsung Galaxy Ace dengan Sistem operasi Android GingerBread (2.3.3)	Game berhasil di-install dan bisa dimainkan	Game berhasil di-install dan tidak bisa dimainkan	TIDAK BERHASIL
2	Game di-install pada perangkat SmartFren i3 dengan Sistem operasi Android Jellybeans (4.3)	Game berhasil di-install dan bisa dimainkan	Game berhasil di-install dan bisa dimainkan	BERHASIL

5. Resolusi layar

Tabel 4.5
Tabel pengujian resolusi layar

No	Jenis Pengujian	Ekspektasi	Output	Ket
1	Game dimainkan pada perangkat dengan resolusi 480x320	Game ditampilkan fullscreen	Game tidak dapat dimainkan, application not responding	TIDAK BERHASIL
2	Game dimainkan pada perangkat dengan resolusi 960x540	Game ditampilkan fullscreen	Game ditampilkan fullscreen	BERHASIL

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Setelah selesai melalui semua tahap pembuatan dan pengembangan Game Arcade Corgi Adventure dari perencanaan, analisa, perancangan, hingga implementasi maka penulis menarik beberapa kesimpulan, yaitu :

1. Pembuatan game menggunakan game engine Unity sudah banyak digunakan. Unity menyediakan komponen scene untuk tempat pembuatan dan perancangan desain game. Media penyimpanan juga sudah disediakan berupa fitur playerperf. Penggunaan C# juga cukup mudah karena syntax-syntax C# mirip dengan bahasa pemrograman Java dan C++. Unity juga sekarang telah bisa menggunakan extension yang dapat kombinasikan dengan menggunakan Visual Studio 2015 agar memudahkan apabila programmer tidak familiar dengan MonoDevelop
2. Interaksi antara player dengan objek lainnya dapat menggunakan *collider component* yang ada pada unity. Komponen ini merupakan sebuah jaring tak terlihat yang mengelilingi bentuk object dan bertanggung jawab dalam mendeteksi tabrakan dengan benda lain. Tidak hanya mendeteksi interaksi tabrakan, component ini juga dapat mendahulukan tabrakan dengan menggunakan *Ray-Casting*. *Ray-Casting* adalah sebuah garis vektor antara dua point dalam ruang 3D yang dapat digunakan dalam mendeteksi *intersection* dengan *collider* pada *game object*. *Ray-Casting* dapat digunakan untuk mengambil informasi yang berguna seperti jarak dari titik awal hingga akhir garis.

5.2 Saran

Produk yang dihasilkan dalam Tugas Akhir ini dapat dikembangkan lagi karena masih banyak hal dari segi teknis pemrograman hingga sistem permainan yang belum lengkap. Adapun saran yang diberikan untuk pengembangan Tugas Akhir ini diantaranya:

1. Meningkatkan segi grafis dan animasi agar game lebih menarik dan hidup.

2. Menambah level dengan tema yang lebih beragam.
3. Menambahkan tingkat kesulitan pada level agar lebih menantang.
4. Menambah item yang dapat dipergunakan oleh pemain.

DAFTAR PUSTAKA

Bell, Douglas.2005. Software Engineering for Students : A Programming Approach. UK : Pearson.

Booch, Grady, dkk, 2007, "Object – Oriented Analysis and Design with Applications 3rd Edition",Pearson Education, Inc.

Bruegge, Bernd & Allen H. Dutoit, 2004, Object – Oriented Software Engineering 2nd Edition, USA: Pearson Education, Inc.

Clark, Dan. 2013. Beginning C# Object-Oriented Programming, 2nd Edition. New York : Apress.

Dennis, Alan, Barbara Haley Wixom, dkk., 2009, "System Analysis Design UML Version 2.0 An Object-Oriented Approach 3rd Edition", John Wiley & Sons Inc.

Kirby, Niel.2011. Introduction to Game AI. USA : Course Technology PTR.

Marsic, Ivan.2012. Software Engineering. USA: Rutgers University.

Mulchorne, Keiran. 2010. An Introduction to Object Oriented Programming with C#. Ireland: University College Cork.

Pitman, Niel. 2005.UML 2.0 in a Nutshell. USA: O'Rielly.

Pressman, Roger. 2001. Software Engineering : A Practitioner's Approach. New York : McGraw-Hill.

Radack, Shirley.2009. The System Software Life Cycle. New York

Rumbaugh, James. 2004. The Unified Modelling Language Reference Manual Second Edition. USA:Pearson Education.

Sibero, Ivan C., 2008, Langkah Mudah Membuat Game 3D, Yogjakarta: Media Kom.

Smith, Matt. 2013. Unity 4.x Cookbook. Birmingham : Packt Publishing.

Wu, Thomas.2010.An Introducing to Object Oriented Programming with Java. New York : McGraw-Hill.l

Wagner, Ferdinand, dkk, 2006, "Modeling Software with Finite State Machines : a Practical Approach". Auerbach Publication.

Whitten, Jeffrey L., Lonnie D. Bentley, 2007, "System Analysis and Design Methods 7th Edition", McGraw-Hill.

Wright, David R, 2005, Finite State Machines, USA: Carolina State University.

http://www.ogcio.gov.hk/en/infrastructure/methodology/oom/oom_introduction.htm, diakses tanggal 18 September jam 20:22 WIB.

LAMPIRAN

LISTING PROGRAM

1. AIMoveOnSight.cs

```
using UnityEngine;
using System.Collections;

public class AIMoveOnSight : MonoBehaviour
{
	public float Speed=3f;
	/// The maximum distance at which the AI can see the player
	 public float ViewDistance = 10f;
	 /// The character is facing right by default
	 public bool CharacterFacingRight = true;
	/// the horizontal distance from the player at which the agent will stop moving.
	 public float StopDistance = 1f;

	private float _canFireIn;
	private Vector2 _direction;
	private float _distance;
	private CorgiController _controller;
	private Animator _animator;
	private int _facingModifier;

	void Start ()
	{
		_controller = GetComponent<CorgiController>();
		_animator = GetComponent<Animator>();
		_direction=Vector2.right;
		 if (CharacterFacingRight)
		_facingModifier = -1;
		else
		 _facingModifier = 1;
	}

	void Update ()
	{
		bool hit=false;
		_distance = 0;
		// we cast a ray to the left of the agent to check for a Player
		Vector2 raycastOrigin = new Vector2(transform.position.x,
		transform.position.y +(transform.localScale.y/2));
		RaycastHit2D raycast = CorgiTools.CorgiRayCast(raycastOrigin,-Vector2.right,ViewDistance,1<<
		LayerMask.NameToLayer("Player"),true,Color.gray);
		// if we see a player
		if (raycast)
		{
			hit=true;
			_direction = -Vector2.right;
			_distance= raycast.distance;
		}

		raycastOrigin = new Vector2(transform.position.x,transform.position.y+
		(transform.localScale.y/2));
		raycast = CorgiTools.CorgiRayCast(raycastOrigin,Vector2.right,ViewDistance,1<<
		LayerMask.NameToLayer("Player"),true,Color.gray);
		if (raycast)
		{
			hit=true;
			 _direction = Vector2.right;
			_distance = raycast.distance;
		}


		// if the ray has hit the player, moving the agent in that direction
		 if ((hit) &&  (_distance > StopDistance))
			_controller.SetHorizontalForce(_direction.x * Speed);
		else
			 _controller.SetHorizontalForce(0);

		if (_direction == Vector2.right)
```

```
                transransform.localScale = new Vector3(-Mathf.Abs(transform.localScale.x) *
                _facingModifier, transform.localScale.y, transform.localScale.z);
                else
                transransform.localScale = new Vector3(Mathf.Abs(transform.localScale.x) *
                _facingModifier, transform.localScale.y, transform.localScale.z);

                if (_animator != null)
                _animator.SetFloat("Speed", Mathf.Abs(_controller.Speed.x));
        }

}
```

2. AIShootOnSight.cs

```
using UnityEngine;
using System.Collections;

public class AIShootOnSight : MonoBehaviour
{

        /// The fire rate (in seconds)
        public float FireRate = 1;
        /// The kind of projectile shot by the agent
        public Projectile Projectile;
        /// The maximum distance at which the AI can shoot at the player
        public float ShootDistance = 10f;

        private float _canFireIn;
        private Vector2 _direction;
        private Vector2 _directionLeft;
        private Vector2 _directionRight;
        private CorgiController _controller;

        void Start ()
        {
                _directionLeft = new Vector2(-1,0);
                _directionRight = new Vector2(1,0);
                _controller = GetComponent<CorgiController>();
        }

        void Update ()
        {
                // fire cooldown
                if ((_canFireIn-=Time.deltaTime) > 0)
                {
                        return;
                }

                // the direction of the AI
                if (transform.localScale.x < 0)
                {
                        _direction=-_directionLeft;
                }
                else
                {
                        _direction=-_directionRight;
                }

                // cast a ray in front of the agent to check for a Player
                Vector2 raycastOrigin = new Vector2(transform.position.x,transform.position.y
                                        -(transform.localScale.y/2));
                RaycastHit2D raycast = Physics2D.Raycast(raycastOrigin,_direction,ShootDistance,1
                                        <<LayerMask.NameToLayer("Player"));
                if (!raycast)
                return;

                // if the ray has hit the player, fire a projectile
                Projectile projectile = (Projectile)Instantiate(Projectile, transform.position,transform.rotation);
                projectile.Initialize(gameObject,_direction,_controller.Speed);
```

```
                    _canFireIn=FireRate;
            }
    }
```

3. AISimpleWalk.cs

```
using UnityEngine;
using System.Collections;
public class AISimpleWalk : MonoBehaviour,IPlayerRespawnListener
{
        public float Speed;
        public bool GoesRightInitially=true;

        private CorgiController _controller;
         private Vector2 _direction;
        private Vector2 _startPosition;
        private Vector2 _initialDirection;

        public void Awake ()
    {
                _controller = GetComponent<CorgiController>();
                _startPosition = transform.position;
                _direction = GoesRightInitially ? Vector2.right : -Vector2.right;
                _initialDirection = _direction;
        }

        public void Update ()
        {

                _controller.SetHorizontalForce(_direction.x * Speed);

                if ((_direction.x < 0 && _controller.State.IsCollidingLeft) ||
                    (_direction.x > 0 && _controller.State.IsCollidingRight))
                {
                        _direction = -_direction;
                        transform.localScale = new Vector3(-transform.localScale.x,
                                                transform.localScale.y,transform.localScale.z);
                }
        }

        public void onPlayerRespawnInThisCheckpoint (CheckPoint checkpoint, CharacterBehavior player)
        {
                _direction = _initialDirection;
                transform.localScale=new Vector3(1,1,1);
                transform.position=_startPosition;
                gameObject.SetActive(true);
        }

}
```

4. CorgiController.cs

```
using UnityEngine;
using System.Collections;
using UnitySampleAssets.CrossPlatformInput;
using System.Collections.Generic;
[RequireComponent(typeof(BoxCollider2D))]

public class CorgiController : MonoBehaviour
{
        public CorgiControllerState State { get; private set; }
        public CorgiControllerParameters DefaultParameters;
        public CorgiControllerParameters Parameters{get{return _overrideParameters ?? DefaultParameters;}}

        public LayerMask PlatformMask=0;
        public LayerMask MovingPlatformMask=0;
        public LayerMask EdgeColliderPlatformMask=0;
        public GameObject StandingOn { get; private set; }
        public Vector2 Speed { get{ return _speed; } }
```

```
public int NumberOfHorizontalRays = 8;
public int NumberOfVerticalRays = 8;
public float RayOffset=0.05f;

public Vector3 ColliderCenter {get
        {
                Vector3 colliderCenter = Vector3.Scale(transform.localScale, _boxCollider.offset);
                return colliderCenter;
        }}
public Vector3 ColliderPosition {get
        {
                Vector3 colliderPosition = transform.position + ColliderCenter;
                return colliderPosition;
        }}
public Vector3 ColliderSize {get
        {
                Vector3 colliderSize = Vector3.Scale(transform.localScale, _boxCollider.size);
                return colliderSize;
        }}
public Vector3 BottomPosition {get
        {
                Vector3 colliderBottom = new Vector3(ColliderPosition.x,ColliderPosition.y -
                                        (ColliderSize.y / 2),ColliderPosition.z);
                return colliderBottom;
        }}

private CorgiControllerParameters _overrideParameters;
private Vector2 _speed;
private float _fallSlowFactor;
private Vector2 _externalForce;
private Vector2 _newPosition;
private Transform _transform;
private BoxCollider2D _boxCollider;
private GameObject _lastStandingOn;
private LayerMask _platformMaskSave;
private float _movingPlatformsCurrentGravity;

private const float _largeValue=500000f;
private const float _smallValue=0.0001f;
private const float _obstacleHeightTolerance=0.05f;
private const float _movingPlatformsGravity=-150;

private Rect _rayBoundsRectangle;

private List<RaycastHit2D> _contactList;

public void Awake()
{
        _transform=transform;
        _boxCollider = (BoxCollider2D)GetComponent<BoxCollider2D>();
        _contactList = new List<RaycastHit2D>();
        State = new CorgiControllerState();

        _platformMaskSave = PlatformMask;
        PlatformMask |= EdgeColliderPlatformMask;
        PlatformMask |= MovingPlatformMask;

        State.Reset();
        SetRaysParameters();
}

public void AddForce(Vector2 force)
{
        _speed += force;
        _externalForce += force;
}
```

```
public void AddHorizontalForce(float x)
{
	_speed.x += x;
	_externalForce.x += x;
}

public void AddVerticalForce(float y)
{
	_speed.y += y;
	_externalForce.y += y;
}

public void SetForce(Vector2 force)
{
	_speed = force;
	_externalForce = force;
}

public void SetHorizontalForce (float x)
{
	_speed.x = x;
	_externalForce.x = x;
}

public void SetVerticalForce (float y)
{
	_speed.y = y;
	_externalForce.y = y;
}

private void Update()
{
	// nothing
}

private void LateUpdate()
{
	_contactList.Clear();

	_speed.y += (Parameters.Gravity + _movingPlatformsCurrentGravity) * Time.deltaTime;

	if (_fallSlowFactor!=0)
	{
		_speed.y*=_fallSlowFactor;
	}

	_newPosition=Speed * Time.deltaTime;

	State.WasGroundedLastFrame = State.IsCollidingBelow;
	State.WasTouchingTheCeilingLastFrame = State.IsCollidingAbove;
	State.Reset();

	SetRaysParameters();

	CastRaysToTheSides();
	CastRaysBelow();
	CastRaysAbove();

	_transform.Translate(_newPosition,Space.World);

	SetRaysParameters();

	if (Time.deltaTime > 0)
		_speed = _newPosition / Time.deltaTime;

	Mathf.Clamp(_speed.x,-Parameters.MaxVelocity.x,Parameters.MaxVelocity.x);
	Mathf.Clamp(_speed.y,-Parameters.MaxVelocity.y,Parameters.MaxVelocity.y);
```

```
            if( !State.WasGroundedLastFrame && State.IsCollidingBelow )
                State.JustGotGrounded=true;

            if (State.IsCollidingLeft || State.IsCollidingRight || State.IsCollidingBelow || State.IsCollidingRight)
            {
                OnCorgiColliderHit();
            }


            _externalForce.x=0;
            _externalForce.y=0;
}

private void CastRaysToTheSides()
{
            float movementDirection=1;
            if ((_speed.x < 0) || (_externalForce.x<0))
                movementDirection = -1;

            float horizontalRayLength = Mathf.Abs(_speed.x*Time.deltaTime) +
                                        _rayBoundsRectangle.width/2 + RayOffset*2;

            Vector2 horizontalRayCastFromBottom=new Vector2(_rayBoundsRectangle.center.x,
                                    _rayBoundsRectangle.yMin+_obstacleHeightTolerance);

            Vector2 horizontalRayCastToTop=new Vector2(          _rayBoundsRectangle.center.x,
                                    _rayBoundsRectangle.yMax-_obstacleHeightTolerance);


            RaycastHit2D[] hitsStorage = new RaycastHit2D[NumberOfHorizontalRays];

            for (int i=0; i<NumberOfHorizontalRays;i++)
            {
                Vector2 rayOriginPoint = Vector2.Lerp(horizontalRayCastFromBottom,
                horizontalRayCastToTop,(float)i/(float)(NumberOfHorizontalRays-1));

                if ( State.WasGroundedLastFrame && i == 0 )
                    hitsStorage[i] = CorgiTools.CorgiRayCast (rayOriginPoint,movementDirection
                                    *Vector2.right,horizontalRayLength,
                                     PlatformMask,true,Color.red);
                else
                    hitsStorage[i] = CorgiTools.CorgiRayCast (rayOriginPoint,movementDirection
                                    *Vector2.right,horizontalRayLength,
                                    PlatformMask & ~EdgeColliderPlatformMask,true,Color.red);

                if (hitsStorage[i].distance >0)
                {
                    float hitAngle = Mathf.Abs(Vector2.Angle(hitsStorage[i].normal, Vector2.up));


                    if (hitAngle > Parameters.MaximumSlopeAngle)
                    {

                        if (movementDirection < 0)
                            State.IsCollidingLeft=true;
                        else
                            State.IsCollidingRight=true;


                        State.SlopeAngleOK=false;

                        if (movementDirection<=0)
                        {
                            _newPosition.x = -Mathf.Abs(hitsStorage[i].point.x
                                - horizontalRayCastFromBottom.x)
                                + _rayBoundsRectangle.width/2
                                + RayOffset*2;
                        }
```

```
                                        else
                                        {
                                                _newPosition.x = Mathf.Abs(hitsStorage[i].point.x -
                                                        horizontalRayCastFromBottom.x)
                                                        - _rayBoundsRectangle.width/2
                                                        - RayOffset*2;

                                        }

                                        _contactList.Add(hitsStorage[i]);
                                        _speed = new Vector2(0, _speed.y);
                                        break;
                                }
                        }
                }

}

private void CastRaysBelow()
{
        if (_newPosition.y < -_smallValue)
        {
                State.IsFalling=true;
        }
        else
        {
                State.IsFalling = false;
        }

        if ((Parameters.Gravity > 0) && (!State.IsFalling))
                return;

        float rayLength = _rayBoundsRectangle.height/2 + RayOffset ;
        if (_newPosition.y<0)
        {
                rayLength+=Mathf.Abs(_newPosition.y);
        }


        Vector2 verticalRayCastFromLeft=new Vector2(_rayBoundsRectangle.xMin+_newPosition.x,
                        _rayBoundsRectangle.center.y+RayOffset);
        Vector2 verticalRayCastToRight=new Vector2(_rayBoundsRectangle.xMax+_newPosition.x,
                        _rayBoundsRectangle.center.y+RayOffset);

        RaycastHit2D[] hitsStorage = new RaycastHit2D[NumberOfVerticalRays];
        float smallestDistance=_largeValue;
        int smallestDistanceIndex=0;
        bool hitConnected=false;

        for (int i=0; i<NumberOfVerticalRays;i++)
        {
                Vector2 rayOriginPoint = Vector2.Lerp(verticalRayCastFromLeft,verticalRayCastToRight,
                                (float)i/(float)(NumberOfVerticalRays-1));

                if ((_newPosition.y>0) && (!State.WasGroundedLastFrame))
                        hitsStorage[i] = CorgiTools.CorgiRayCast (rayOriginPoint,-
                                        Vector2.up,rayLength,PlatformMask
                                        & ~EdgeColliderPlatformMask,true,Color.blue);
                else
                        hitsStorage[i] = CorgiTools.CorgiRayCast (rayOriginPoint,-Vector2.up,
                                        rayLength,PlatformMask,true,Color.blue);

                if ((Mathf.Abs(hitsStorage[smallestDistanceIndex].point.y -
                                                        verticalRayCastFromLeft.y)) <  _smallValue)
                {
                        break;
                }
```

```
                if (hitsStorage[i])
                {
                        hitConnected=true;
                        if (hitsStorage[i].distance<smallestDistance)
                        {
                                smallestDistanceIndex=i;
                                smallestDistance = hitsStorage[i].distance;
                        }
                }
        }
        if (hitConnected)
        {

                StandingOn=hitsStorage[smallestDistanceIndex].collider.gameObject;

                //_contactList.Add(hitsStorage[smallestDistanceIndex]);

                if (!State.WasGroundedLastFrame && smallestDistance < _rayBoundsRectangle.size.y/2
                        && StandingOn.layer==LayerMask.NameToLayer("OneWayPlatforms"))
                {
                        State.IsCollidingBelow=false;
                        return;
                }

                State.IsFalling=false;
                State.IsCollidingBelow=true;

                _newPosition.y = -Mathf.Abs(hitsStorage[smallestDistanceIndex].point.y
                                                - verticalRayCastFromLeft.y)
                                                + _rayBoundsRectangle.height/2
                                                + RayOffset;

                if (_externalForce.y>0)
                {
                        _newPosition.y += _speed.y * Time.deltaTime;
                        State.IsCollidingBelow = false;
                }

                if (!State.WasGroundedLastFrame && _speed.y>0)
                {
                        _newPosition.y += _speed.y * Time.deltaTime;
                }



                if (Mathf.Abs(_newPosition.y)<_smallValue)
                        _newPosition.y = 0;

                // check if the character is standing on a moving platform
                PathFollow movingPlatform = hitsStorage[smallestDistanceIndex].
                                                collider.GetComponent<PathFollow>();
                State.OnAMovingPlatform=false;
                if (movingPlatform!=null)
                {
                        _movingPlatformsCurrentGravity=_movingPlatformsGravity;
                        State.OnAMovingPlatform=true;
                        _transform.Translate(movingPlatform.CurrentSpeed*Time.deltaTime);
                        _newPosition.y = 0;
                }
                else
                {
                        _movingPlatformsCurrentGravity=0;
                }
        }
        else
        {
                _movingPlatformsCurrentGravity=0;
                State.IsCollidingBelow=false;
```

```
        }
}

/// If we're in the air and moving up, cast rays above the character's head to check for collisions
private void CastRaysAbove()
{

        if (_newPosition.y<0)
                return;

        float rayLength = State.IsGrounded?RayOffset : _newPosition.y*Time.deltaTime;
        rayLength+=_rayBoundsRectangle.height/2;

        bool hitConnected=false;

        Vector2 verticalRayCastStart=new Vector2(_rayBoundsRectangle.xMin+_newPosition.x,
                        _rayBoundsRectangle.center.y);
        Vector2 verticalRayCastEnd=new Vector2(    _rayBoundsRectangle.xMax+_newPosition.x,
                        _rayBoundsRectangle.center.y);

        RaycastHit2D[] hitsStorage = new RaycastHit2D[NumberOfVerticalRays];
        float smallestDistance=_largeValue;

        for (int i=0; i<NumberOfVerticalRays;i++)
        {
                Vector2 rayOriginPoint = Vector2.Lerp(verticalRayCastStart,verticalRayCastEnd,
                                        (float)i/(float)(NumberOfVerticalRays-1));
                hitsStorage[i] = CorgiTools.CorgiRayCast
                                (rayOriginPoint,Vector2.up,rayLength,PlatformMask
                                        & ~EdgeColliderPlatformMask,true,Color.green);

                if (hitsStorage[i])
                {
                        hitConnected=true;
                        if (hitsStorage[i].distance<smallestDistance)
                        {
                                smallestDistance = hitsStorage[i].distance;
                        }
                }
        }

        if (hitConnected)
        {
                _speed.y=0;
                _newPosition.y = smallestDistance - _rayBoundsRectangle.height/2   ;

                if ( (State.IsGrounded) && (_newPosition.y<0) )
                {
                        _newPosition.y=0;
                }

                State.IsCollidingAbove=true;

                if (!State.WasTouchingTheCeilingLastFrame)
                {
                        _newPosition.x=0;
                        _speed = new Vector2(0, _speed.y);
                }
        }
}

/// Creates a rectangle with the boxcollider's size for ease of use
public void SetRaysParameters()
{

        _rayBoundsRectangle = new Rect(_boxCollider.bounds.min.x,
                        _boxCollider.bounds.min.y,
                        _boxCollider.bounds.size.x,
```

```
                                _boxCollider.bounds.size.y);

        Debug.DrawLine(new Vector2(_rayBoundsRectangle.center.x,_rayBoundsRectangle.yMin),
                                new Vector2(_rayBoundsRectangle.center.x,_rayBoundsRectangle.yMax));
        Debug.DrawLine(new Vector2(_rayBoundsRectangle.xMin,_rayBoundsRectangle.center.y),
                                new Vector2(_rayBoundsRectangle.xMax,_rayBoundsRectangle.center.y));


}

/// Disables the collisions for the specified duration
public IEnumerator DisableCollisions(float duration)
{
        CollisionsOff();
        yield return new WaitForSeconds (duration);
        CollisionsOn();
}

public void ResetMovingPlatformsGravity()
{
        _movingPlatformsCurrentGravity=0f;
}

public void CollisionsOn()
{
        PlatformMask=_platformMaskSave;
        PlatformMask |= EdgeColliderPlatformMask;
        PlatformMask |= MovingPlatformMask;
}

public void CollisionsOff()
{
        PlatformMask=0;
}

public void ResetParameters()
{
        _overrideParameters = DefaultParameters;
}

public void SlowFall(float factor)
{
        _fallSlowFactor=factor;
}

/// triggered when the character's raycasts collide with something
private void OnCorgiColliderHit()
{
        foreach (RaycastHit2D hit in _contactList )
        {
                Rigidbody2D body = hit.collider.attachedRigidbody;
                if (body == null || body.isKinematic)
                        return;

                Vector3 pushDir = new Vector3(_externalForce.x, 0, 0);

                body.velocity = pushDir.normalized * Parameters.Physics2DPushForce;
        }
}

/// triggered when the character enters a collider
public void OnTriggerEnter2D(Collider2D collider)
{

        CorgiControllerPhysicsVolume2D parameters = collider.gameObject
                                .GetComponent<CorgiControllerPhysicsVolume2D>();
        if (parameters == null)
```

```
                return;

            _overrideParameters = parameters.ControllerParameters;
        }

        /// triggered while the character stays inside another collider
        public void OnTriggerStay2D( Collider2D collider )
        {
        }

        /// triggered when the character exits a collider
        public void OnTriggerExit2D(Collider2D collider)
        {
            CorgiControllerPhysicsVolume2D parameters = collider.gameObject
                                                        .GetComponent<CorgiControllerPhysicsVolume2D>();
            if (parameters == null)
                return;

            // if the object we were colliding with had parameters, we reset our character's parameters
            _overrideParameters = null;
        }
    }
```

5. CorgiControllerParameters.cs

```
    using System;
    using UnityEngine;
    using System.Collections;
    [RequireComponent(typeof(Collider2D))]


    [Serializable]
    public class CorgiControllerParameters
    {
        /// Maximum velocity for your character
        public Vector2 MaxVelocity = new Vector2(200f, 200f);
        public int MaxSpeed=200;
        /// Maximum angle (in degrees) the character can walk on
        [Range(0,90)]
        public float MaximumSlopeAngle = 45;
        public float Gravity = -15;
        // Speed factor on the ground
        public float SpeedAccelerationOnGround = 20f;
        // Speed factor in the air
        public float SpeedAccelerationInAir = 5f;

    }
```

6. CharacterBehaviour.cs

```
    using UnityEngine;
    using System.Collections;
    using UnitySampleAssets.CrossPlatformInput;
    public class CharacterBehavior : MonoBehaviour,CanTakeDamage
    {
        public BoxCollider2D HeadCollider ;

        public int Health {get; set; }

        public CharacterBehaviorState BehaviorState { get; private set; }
        public CharacterBehaviorParameters DefaultBehaviorParameters;
        public CharacterBehaviorParameters BehaviorParameters{get{return _overrideBehaviorParameters ??
        DefaultBehaviorParameters;}}
        public CharacterBehaviorPermissions Permissions ;

        public ParticleSystem TouchTheGroundEffect;
        public ParticleSystem HurtEffect;

        public AudioClip PlayerJumpSfx;
        public AudioClip PlayerHitSfx;
```

```
public bool JumpAuthorized
{
	get
	{
		if ( (BehaviorParameters.JumpRestrictions ==
			CharacterBehaviorParameters.JumpBehavior.CanJumpAnywhere)
		 ||  (BehaviorParameters.JumpRestrictions ==
			CharacterBehaviorParameters.JumpBehavior.
				CanJumpAnywhereAnyNumberOfTimes) )
			return true;

		if (BehaviorParameters.JumpRestrictions ==
			CharacterBehaviorParameters.JumpBehavior.CanJumpOnGround)
				return _controller.State.IsGrounded;

		return false;
	}
}

private CameraController _sceneCamera;
protected CorgiController _controller;

private Animator _animator;
private CharacterJetpack _jetpack;
private CharacterShoot _shoot;
private Color _initialColor;

private CharacterBehaviorParameters _overrideBehaviorParameters;
private float _originalGravity;

private float _normalizedHorizontalSpeed;

private float _jumpButtonPressTime = 0;
private bool _jumpButtonPressed=false;
private bool _jumpButtonReleased=false;

private bool _isFacingRight=true;

private float _horizontalMove;
private float _verticalMove;

void Awake()
{
	BehaviorState = new CharacterBehaviorState();
	_sceneCamera = GameObject.FindGameObjectWithTag("MainCamera").
			GetComponent<CameraController>();
	_controller = GetComponent<CorgiController>();
	_jetpack = GetComponent<CharacterJetpack>();
	_shoot = GetComponent<CharacterShoot> ();
	Health=BehaviorParameters.MaxHealth;

	if (GetComponent<Renderer>()!=null)
		_initialColor=GetComponent<Renderer>().material.color;
}

public virtual void Start()
{
	_animator = GetComponent<Animator>();
	_isFacingRight = transform.localScale.x > 0;

	_originalGravity = _controller.Parameters.Gravity;

	BehaviorState.Initialize();
	BehaviorState.NumberOfJumpsLeft=BehaviorParameters.NumberOfJumps;

	BehaviorState.CanJump=true;
}
```

```
protected virtual void Update()
{
	if (BehaviorParameters.UseDefaultMecanim)
	{
		UpdateAnimator ();
	}

	if (!BehaviorState.IsDead)
	{
		GravityActive(true);

		HorizontalMovement();
		VerticalMovement();

		BehaviorState.CanShoot=true;

		ClimbLadder();
		WallClinging ();

		if (BehaviorState.Dashing)
		{
			GravityActive(false);
			_controller.SetVerticalForce(0);
		}

		if (!BehaviorState.Firing)
		{
			BehaviorState.FiringStop=false;
		}

		if (JumpAuthorized)
		{
			if ( (_jumpButtonPressTime!=0)
			  && (Time.time - _jumpButtonPressTime >=
				BehaviorParameters.JumpMinimumAirTime)
			  && (_controller.Speed.y >
				Mathf.Sqrt(Mathf.Abs(_controller.Parameters.Gravity)))
			  && (_jumpButtonReleased)
			  && (!_jumpButtonPressed||BehaviorState.Jetpacking))
			{
				_jumpButtonReleased=false;
				if (BehaviorParameters.JumpIsProportionalToThePressTime)

					_controller.AddForce(new Vector2(0,12 *
						Mathf.Abs(_controller.Parameters.Gravity)
							* Time.deltaTime ));
			}
		}
	}
	else
	{
		_controller.SetHorizontalForce(0);
	}
}

void LateUpdate()
{
	if (_controller.State.JustGotGrounded)
	{
		BehaviorState.NumberOfJumpsLeft=BehaviorParameters.NumberOfJumps;
	}

}
```

```
private void UpdateAnimator()
{

        CorgiTools.UpdateAnimatorBool(_animator,"Grounded",_controller.State.IsGrounded);
        CorgiTools.UpdateAnimatorFloat(_animator,"Speed",Mathf.Abs(_controller.Speed.x));
        CorgiTools.UpdateAnimatorFloat(_animator,"vSpeed",_controller.Speed.y);
        CorgiTools.UpdateAnimatorBool(_animator,"Running",BehaviorState.Running);
        CorgiTools.UpdateAnimatorBool(_animator,"Dashing",BehaviorState.Dashing);
        CorgiTools.UpdateAnimatorBool(_animator,"Crouching",BehaviorState.Crouching);
        CorgiTools.UpdateAnimatorBool(_animator,"LookingUp",BehaviorState.LookingUp);
        CorgiTools.UpdateAnimatorBool(_animator,"WallClinging",BehaviorState.WallClinging);
        CorgiTools.UpdateAnimatorBool(_animator,"Jetpacking",BehaviorState.Jetpacking);
        CorgiTools.UpdateAnimatorBool(_animator,"Diving",BehaviorState.Diving);
        CorgiTools.UpdateAnimatorBool(_animator,"LadderClimbing",BehaviorState.LadderClimbing);
        CorgiTools.UpdateAnimatorFloat(_animator,"LadderClimbingSpeed",
                                                        BehaviorState.LadderClimbingSpeed);
        CorgiTools.UpdateAnimatorBool(_animator,"FiringStop",BehaviorState.FiringStop);
        CorgiTools.UpdateAnimatorBool(_animator,"Firing",BehaviorState.Firing);
        CorgiTools.UpdateAnimatorInteger(_animator,"FiringDirection",BehaviorState.FiringDirection);
        CorgiTools.UpdateAnimatorBool(_animator,"MeleeAttacking",BehaviorState.MeleeAttacking);
}


public void SetHorizontalMove(float value)
{
        _horizontalMove=value;
}

public void SetVerticalMove(float value)
{
        _verticalMove=value;
}

private void HorizontalMovement()
{
        if (!BehaviorState.CanMoveFreely)
                return;

        if (_horizontalMove>0.1f)
        {
                _normalizedHorizontalSpeed = _horizontalMove;
                if (!_isFacingRight)
                        Flip();
        }
        else if (_horizontalMove<-0.1f)
        {
                _normalizedHorizontalSpeed = _horizontalMove;
                if (_isFacingRight)
                        Flip();
        }
        else
        {
                _normalizedHorizontalSpeed=0;
        }

        var movementFactor = _controller.State.IsGrounded ?
                                _controller.Parameters.SpeedAccelerationOnGround :
                                _controller.Parameters.SpeedAccelerationInAir;
        if (BehaviorParameters.SmoothMovement)
                _controller.SetHorizontalForce(Mathf.Lerp(_controller.Speed.x,
                        _normalizedHorizontalSpeed * BehaviorParameters.MovementSpeed,
                                Time.deltaTime * movementFactor));
        else
                _controller.SetHorizontalForce(_normalizedHorizontalSpeed *
                                                        BehaviorParameters.MovementSpeed);
}
```

```
private void VerticalMovement()
{
	if ( (_verticalMove>0) && (_controller.State.IsGrounded) )
	{
		BehaviorState.LookingUp = true;
		_sceneCamera.LookUp();
	}
	else
	{
		BehaviorState.LookingUp = false;
		_sceneCamera.ResetLookUpDown();
	}

	if (_controller.State.JustGotGrounded)
	{
		Instantiate(TouchTheGroundEffect,_controller.BottomPosition,transform.rotation);
	}

	if (!BehaviorState.CanMoveFreely)
		return;

	if ( (_verticalMove<-0.1) && (_controller.State.IsGrounded) && (Permissions.CrouchEnabled) )
	{
		BehaviorState.Crouching = true;
		BehaviorParameters.MovementSpeed = BehaviorParameters.CrouchSpeed;
		BehaviorState.Running=false;
		_sceneCamera.LookDown();
	}
	else
	{
		if (BehaviorState.Crouching)
		{
			if (HeadCollider==null)
			{
				BehaviorState.Crouching=false;
				return;
			}
			bool headCheck = Physics2D.OverlapCircle(HeadCollider.transform.position,
					HeadCollider.size.x/2,_controller.PlatformMask);

			if (!headCheck)
			{
				if (!BehaviorState.Running)
				BehaviorParameters.MovementSpeed =
						BehaviorParameters.WalkSpeed;
				BehaviorState.Crouching = false;
				BehaviorState.CanJump=true;
			}
			else
			{

				BehaviorState.CanJump=false;
			}
		}
	}

	if (BehaviorState.CrouchingPreviously!=BehaviorState.Crouching)
	{
		Invoke ("RecalculateRays",Time.deltaTime*10);
	}

	BehaviorState.CrouchingPreviously=BehaviorState.Crouching;

}
```

```
public void RecalculateRays()
{
	_controller.SetRaysParameters();
}

public void RunStart()
{
	if (!Permissions.RunEnabled)
		return;

		if (!BehaviorState.CanMoveFreely)
		return;

	if (_controller.State.IsGrounded && !BehaviorState.Crouching)
	{
		BehaviorParameters.MovementSpeed = BehaviorParameters.RunSpeed;
		BehaviorState.Running=true;
	}
}

public void RunStop()
{
	BehaviorParameters.MovementSpeed = BehaviorParameters.WalkSpeed;
	BehaviorState.Running=false;
}

public void JumpStart()
{

	if (!Permissions.JumpEnabled  || !JumpAuthorized || BehaviorState.IsDead
                                                    || _controller.State.IsCollidingAbove)
		return;

	if (_controller.State.IsGrounded
	   || BehaviorState.LadderClimbing
	   || BehaviorState.WallClinging
	   || BehaviorState.NumberOfJumpsLeft>0)
		BehaviorState.CanJump=true;
	else
		BehaviorState.CanJump=false;

	if ( (!BehaviorState.CanJump) && !(BehaviorParameters.JumpRestrictions==
		CharacterBehaviorParameters.JumpBehavior.CanJumpAnywhereAnyNumberOfTimes) )
		return;

	if (_verticalMove<0 && _controller.State.IsGrounded)
	{
		if (_controller.StandingOn.layer==LayerMask.NameToLayer("OneWayPlatforms"))
		{
			_controller.transform.position=new
                                Vector2(transform.position.x,transform.position.y-0.1f);
			StartCoroutine(_controller.DisableCollisions(0.3f));
			_controller.ResetMovingPlatformsGravity();
			return;
		}
	}

	if (_verticalMove>=0 && _controller.State.IsGrounded)
	{
		if (_controller.StandingOn.layer==LayerMask.NameToLayer("MovingPlatforms"))
		{
			StartCoroutine(_controller.DisableCollisions(0.3f));
			_controller.ResetMovingPlatformsGravity();
		}
	}
```

```
                BehaviorState.NumberOfJumpsLeft=BehaviorState.NumberOfJumpsLeft-1;
                BehaviorState.LadderClimbing=false;
                BehaviorState.CanMoveFreely=true;
                GravityActive(true);

                _jumpButtonPressTime=Time.time;
                _jumpButtonPressed=true;
                _jumpButtonReleased=false;

                _controller.SetVerticalForce(Mathf.Sqrt( 2f * BehaviorParameters.JumpHeight *
                                                        Mathf.Abs(_controller.Parameters.Gravity) ));

                if (PlayerJumpSfx!=null)
                        SoundManager.Instance.PlaySound(PlayerJumpSfx,transform.position);

                float wallJumpDirection;
                if (BehaviorState.WallClinging)
                {

                        if (_controller.State.IsCollidingRight)
                        {
                                wallJumpDirection=-1f;
                        }
                        else
                        {
                                wallJumpDirection=1f;
                        }
                        _controller.SetForce(new
                                Vector2(wallJumpDirection*BehaviorParameters.WallJumpForce,Mathf.Sqrt(
                                2f * BehaviorParameters.JumpHeight *
                                                        Mathf.Abs(_controller.Parameters.Gravity) )));
                        BehaviorState.WallClinging=false;
                }
        }

        public void JumpStop()
        {
                _jumpButtonPressed=false;
                _jumpButtonReleased=true;
        }

        void ClimbLadder()
        {
                if (BehaviorState.LadderColliding)
                {
                        if (_verticalMove>0.1 && !BehaviorState.LadderClimbing &&
                                                                !BehaviorState.LadderTopColliding
                                                                        && !BehaviorState.Jetpacking)
                        {
                                BehaviorState.LadderClimbing=true;
                                _controller.CollisionsOn();

                                BehaviorState.CanMoveFreely=false;
                                if (_shoot!=null)
                                        _shoot.ShootStop();
                                BehaviorState.LadderClimbingSpeed=0;
                                _controller.SetHorizontalForce(0);
                                _controller.SetVerticalForce(0);
                                GravityActive(false);
                        }

                        if (BehaviorState.LadderClimbing)
                        {
                                BehaviorState.CanShoot=false;
                                GravityActive(false);

                                if (!BehaviorState.LadderTopColliding)
                                        _controller.CollisionsOn();
```

```
                    _controller.SetVerticalForce(_verticalMove *
                                                    BehaviorParameters.LadderSpeed);
                    BehaviorState.LadderClimbingSpeed=Mathf.Abs(_verticalMove);

                }

                if (!BehaviorState.LadderTopColliding)
                {
                    _controller.CollisionsOn();
                }

                if (BehaviorState.LadderClimbing && _controller.State.IsGrounded &&
                                                    !BehaviorState.LadderTopColliding)
                {
                    BehaviorState.LadderColliding=false;
                    BehaviorState.LadderClimbing=false;
                    BehaviorState.CanMoveFreely=true;
                    BehaviorState.LadderClimbingSpeed=0;
                    GravityActive(true);
                }
            }

            if (BehaviorState.LadderTopColliding && _verticalMove<-0.1 && !BehaviorState.LadderClimbing &&
                                                                    _controller.State.IsGrounded)
            {
                _controller.CollisionsOff();
                transform.position=new Vector2(transform.position.x,transform.position.y-0.1f);
                BehaviorState.LadderClimbing=true;
                BehaviorState.CanMoveFreely=false;
                BehaviorState.LadderClimbingSpeed=0;
                _controller.SetHorizontalForce(0);
                _controller.SetVerticalForce(0);
                GravityActive(false);
            }
        }

        public void Dash()
        {
            float _dashDirection;
            float _boostForce;

            if (!Permissions.DashEnabled || BehaviorState.IsDead)
                return;

            if (!BehaviorState.CanMoveFreely)
                return;

            if (_verticalMove>-0.8)
            {
                if (BehaviorState.CanDash)
                {
                    BehaviorState.Dashing=true;

                    if (_isFacingRight) { _dashDirection=1f; } else { _dashDirection = -1f; }
                    _boostForce=_dashDirection*BehaviorParameters.DashForce;
                    BehaviorState.CanDash = false;

                    StartCoroutine(Boost(BehaviorParameters.
                                                    DashDuration,_boostForce,0,"dash") );
                }
            }

            if (_verticalMove<-0.8)
            {
                _controller.CollisionsOn();
                StartCoroutine(Dive());
            }
```

```
}

IEnumerator Boost(float boostDuration, float boostForceX, float boostForceY, string name)
{
	float time = 0f;

	while(boostDuration > time)
	{
		if (boostForceX!=0)
		{
			_controller.AddForce(new Vector2(boostForceX,0));
		}
		if (boostForceY!=0)
		{
			_controller.AddForce(new Vector2(0,boostForceY));
		}
		time+=Time.deltaTime;
		yield return 0;
	}

	if (name=="dash")
	{
		BehaviorState.Dashing=false;
		GravityActive(true);
		yield return new WaitForSeconds(BehaviorParameters.DashCooldown);
		BehaviorState.CanDash = true;
	}
	if (name=="wallJump")
	{
		//nothing
	}
}

IEnumerator Dive()
{
	// Shake parameters : intensity, duration (in seconds) and decay
	Vector3 ShakeParameters = new Vector3(1.5f,0.5f,1f);
	BehaviorState.Diving=true;

	while (!_controller.State.IsGrounded)
	{
		_controller.SetVerticalForce(-Mathf.Abs(_controller.Parameters.Gravity)*2);
		yield return 0;
	}

	_sceneCamera.Shake(ShakeParameters);
	BehaviorState.Diving=false;
}

private void WallClinging()
{
	if (!Permissions.WallClingingEnabled)
		return;

	if (!_controller.State.IsCollidingLeft && !_controller.State.IsCollidingRight)
	{
		BehaviorState.WallClinging=false;
	}

	if (!BehaviorState.CanMoveFreely)
		return;

	if((!_controller.State.IsGrounded) && ( ( (_controller.State.IsCollidingRight) &&
		 (_horizontalMove>0.1f) )	||	( (_controller.State.IsCollidingLeft) &&
							(_horizontalMove<-0.1f) )	))
	{
		if (_controller.Speed.y<0)
		{
```

```
                                BehaviorState.WallClinging=true;
                                _controller.SlowFall(BehaviorParameters.WallClingingSlowFactor);
                        }
                }
                else
                {
                        BehaviorState.WallClinging=false;
                        _controller.SlowFall(0f);
                }
}

private void GravityActive(bool state)
{
        if (state==true)
        {
                if (_controller.Parameters.Gravity==0)
                {
                        _controller.Parameters.Gravity = _originalGravity;
                }
        }
        else
        {
                if (_controller.Parameters.Gravity!=0)
                        _originalGravity = _controller.Parameters.Gravity;
                _controller.Parameters.Gravity = 0;
        }
}

IEnumerator Flicker(Color initialColor, Color flickerColor, float flickerSpeed)
{
        if (GetComponent<Renderer>()!=null)
        {
                for(var n = 0; n < 10; n++)
                {
                        GetComponent<Renderer>().material.color = initialColor;
                        yield return new WaitForSeconds (flickerSpeed);
                        GetComponent<Renderer>().material.color = flickerColor;
                        yield return new WaitForSeconds (flickerSpeed);
                }
                GetComponent<Renderer>().material.color = initialColor;
        }
}

IEnumerator ResetLayerCollision(float delay)
{
        yield return new WaitForSeconds (delay);
        Physics2D.IgnoreLayerCollision(9,12,false);
        Physics2D.IgnoreLayerCollision(9,13,false);
}

public void Kill()
{
        _controller.CollisionsOff();
        GetComponent<Collider2D>().enabled=false;
        BehaviorState.IsDead=true;
        Health=0;
        _controller.ResetParameters();
        ResetParameters();
        _controller.SetForce(new Vector2(0,10));
}

public void Disable()
{
        enabled=false;
        _controller.enabled=false;
        GetComponent<Collider2D>().enabled=false;
}
```

```
public void RespawnAt(Transform spawnPoint)
{
	if(!_isFacingRight)
	{
		Flip ();
	}

	BehaviorState.IsDead=false;
	GetComponent<Collider2D>().enabled=true;
	_controller.CollisionsOn();
	transform.position=spawnPoint.position;
	Health=BehaviorParameters.MaxHealth;
}

public virtual void TakeDamage(int damage,GameObject instigator)
{
	if (PlayerHitSfx!=null)
		SoundManager.Instance.PlaySound(PlayerHitSfx,transform.position);

	Instantiate(HurtEffect,transform.position,transform.rotation);
	Physics2D.IgnoreLayerCollision(9,12,true);
	Physics2D.IgnoreLayerCollision(9,13,true);
	StartCoroutine(ResetLayerCollision(0.5f));

	if (GetComponent<Renderer>()!=null)
	{
		Color flickerColor = new Color32(255, 20, 20, 255);
		StartCoroutine(Flicker(_initialColor,flickerColor,0.05f));
	}

	Health -= damage;
	if (Health<=0)
	{
		LevelManager.Instance.KillPlayer();
	}
}

public void GiveHealth(int health,GameObject instigator)
{
	Health = Mathf.Min (Health + health,BehaviorParameters.MaxHealth);
}

protected virtual void Flip()
{
	transform.localScale = new Vector3(-
				transform.localScale.x,transform.localScale.y
							,transform.localScale.z);
	_isFacingRight = transform.localScale.x > 0;

	if (_jetpack!=null)
	{
		if (_jetpack.Jetpack!=null)
			_jetpack.Jetpack.transform.eulerAngles = new
				Vector3(_jetpack.Jetpack.transform.eulerAngles.x,
					_jetpack.Jetpack.transform.eulerAngles.y+
					180,_jetpack.Jetpack.transform.eulerAngles.z);
	}
	if (_shoot != null)
	{
		_shoot.Flip();
	}
}


public void ResetParameters()
{
	_overrideBehaviorParameters = DefaultBehaviorParameters;
}
```

```
	public void OnTriggerEnter2D(Collider2D collider)
	{

		var parameters = collider.gameObject.GetComponent<CorgiControllerPhysicsVolume2D>();
		if (parameters == null)
			return;
		_overrideBehaviorParameters = parameters.BehaviorParameters;
	}

	public void OnTriggerStay2D( Collider2D collider )
	{
	}

	public void OnTriggerExit2D(Collider2D collider)
	{

		var parameters = collider.gameObject.GetComponent<CorgiControllerPhysicsVolume2D>();
		if (parameters == null)
			return;

		_overrideBehaviorParameters = null;
	}
}
```

7. CharacterBehaviourParameters.cs

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

[Serializable]
public class CharacterBehaviorParameters
{
	/// If set to true, acceleration / deceleration will take place when moving / stopping
	public bool SmoothMovement=true;

	/// Set this to false if you want to implement your own animation system
	public bool UseDefaultMecanim = true;

	/// defines how high the character can jump
	public float JumpHeight = 3.025f;
	/// the minimum time in the air allowed when jumping - this is used for pressure controlled jumps
	public float JumpMinimumAirTime = 0.1f;
	/// the maximum number of jumps allowed (0 : no jump, 1 : normal jump, 2 : double jump, etc...)
	public int NumberOfJumps=3;
	public enum JumpBehavior
	{
		CanJumpOnGround,
		CanJumpAnywhere,
		CantJump,
		CanJumpAnywhereAnyNumberOfTimes
	}
	/// basic rules for jumps : where can the player jump ?
	public JumpBehavior JumpRestrictions;
	/// if true, the jump duration/height will be proportional to the duration of the button's press
	public bool JumpIsProportionalToThePressTime=true;

	/// basic movement speed
	public float MovementSpeed = 8f;
	/// the speed of the character when it's crouching
	public float CrouchSpeed = 4f;
	/// the speed of the character when it's walking
	public float WalkSpeed = 8f;
	/// the speed of the character when it's running
	public float RunSpeed = 16f;
	/// the speed of the character when climbing a ladder
	public float LadderSpeed = 2f;
```

```
        /// the duration of dash (in seconds)
        public float DashDuration = 0.15f;
        /// the force of the dash
        public float DashForce = 5f;
        /// the duration of the cooldown between 2 dashes (in seconds)
        public float DashCooldown = 2f;

        /// the force of a walljump
        public float WallJumpForce = 3f;
        /// the slow factor when wall clinging
        public float WallClingingSlowFactor=0.6f;

        /// the maximum health of the character
        public int MaxHealth = 100;
}
```

8. CharacterBehaviourPermissions.cs

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

[Serializable]
public class CharacterBehaviorPermissions
{

        public bool RunEnabled=true;
        public bool DashEnabled=true;
        public bool JetpackEnabled=true;
        public bool JumpEnabled=true;
        public bool CrouchEnabled=true;
        public bool ShootEnabled=true;
        public bool WallJumpEnabled=true;
        public bool WallClingingEnabled=true;
}
```

9. CharacterShoot.cs

```
using UnityEngine;
using System.Collections;

public class CharacterShoot : MonoBehaviour
{
        public Weapon InitialWeapon;

        public Transform WeaponAttachment;
        public bool EightDirectionShooting=true;
        public bool StrictEightDirectionShooting=true;


        private Weapon _weapon;
        private float _fireTimer;

        private float _horizontalMove;
        private float _verticalMove;

        private CharacterBehavior _characterBehavior;
        private CorgiController _controller;

        void Start ()
        {
                _characterBehavior = GetComponent<CharacterBehavior>();
                _controller = GetComponent<CorgiController>();

                if (WeaponAttachment==null)
                        WeaponAttachment=transform;

                ChangeWeapon(InitialWeapon);
        }
```

```
public void ShootOnce()
{
	if (!_characterBehavior.Permissions.ShootEnabled || _characterBehavior.BehaviorState.IsDead)
		return;
	if (!_characterBehavior.BehaviorState.CanShoot)
	{
		_characterBehavior.BehaviorState.FiringDirection=3;
		return;
	}

	if (!_characterBehavior.BehaviorState.CanMoveFreely)
		return;

	FireProjectile();
	_fireTimer = 0;
}

public void ShootStart()
{
	if (!_characterBehavior.Permissions.ShootEnabled || _characterBehavior.BehaviorState.IsDead)
		return;
	if (!_characterBehavior.BehaviorState.CanShoot)
	{
		_characterBehavior.BehaviorState.FiringDirection=3;
		return;
	}

	if (!_characterBehavior.BehaviorState.CanMoveFreely)
		return;

	_characterBehavior.BehaviorState.FiringStop = false;
	_characterBehavior.BehaviorState.Firing = true;

	_weapon.SetGunFlamesEmission(true);
	_weapon.SetGunShellsEmission (true);
	_fireTimer += Time.deltaTime;
	if(_fireTimer > _weapon.FireRate)
	{
		FireProjectile();
		_fireTimer = 0; // reset timer for fire rate
	}
}

public void ShootStop()
{
	if (!_characterBehavior.Permissions.ShootEnabled)
		return;
	if (!_characterBehavior.BehaviorState.CanShoot)
	{
		_characterBehavior.BehaviorState.FiringDirection=3;
		return;
	}
	_characterBehavior.BehaviorState.FiringStop = true;
	_characterBehavior.BehaviorState.Firing = false;
	_characterBehavior.BehaviorState.FiringDirection=3;
	_weapon.GunFlames.enableEmission=false;
	_weapon.GunShells.enableEmission=false;
}

public void ChangeWeapon(Weapon newWeapon)
{
	if(_weapon!=null)
	{
		ShootStop();
	}
		_weapon=(Weapon)Instantiate(newWeapon,WeaponAttachment.transform.position,
			WeaponAttachment.transform.rotation);
```

```
        _weapon.transform.parent = transform;
        _weapon.SetGunFlamesEmission (false);
        _weapon.SetGunShellsEmission (false);
}

void FireProjectile ()
{
        float HorizontalShoot = _horizontalMove;
        float VerticalShoot = _verticalMove;

        if (_weapon.ProjectileFireLocation==null)
                return;

        if (!EightDirectionShooting)
        {
                HorizontalShoot=0;
                VerticalShoot=0;
        }

        if (StrictEightDirectionShooting)
        {
                HorizontalShoot = Mathf.Round(HorizontalShoot);
                VerticalShoot = Mathf.Round(VerticalShoot);
        }

        float angle = Mathf.Atan2(HorizontalShoot, VerticalShoot) * Mathf.Rad2Deg;

        Vector2 direction = Vector2.up;

        // if the player is not pressing any direction, set the shoot direction based on the direction facing.
        if (HorizontalShoot>-0.1f && HorizontalShoot<0.1f && VerticalShoot>-0.1f && VerticalShoot<0.1f )
        {
                bool _isFacingRight = transform.localScale.x > 0;
                angle=_isFacingRight?90f : -90f;

        }

        // if shooting up
        if ( Mathf.Abs(HorizontalShoot)<0.1f && VerticalShoot>0.1f )
                _characterBehavior.BehaviorState.FiringDirection=1;
        // if shooting diagonal up
        if ( Mathf.Abs(HorizontalShoot)>0.1f && VerticalShoot>0.1f )
                _characterBehavior.BehaviorState.FiringDirection=2;
        // if shooting diagonal down
        if ( Mathf.Abs(HorizontalShoot)>0.1f && VerticalShoot<-0.1f )
                _characterBehavior.BehaviorState.FiringDirection=4;
        // if shooting down
        if ( Mathf.Abs(HorizontalShoot)<0.1f && VerticalShoot<-0.1f )
                _characterBehavior.BehaviorState.FiringDirection=5;
        if (Mathf.Abs(VerticalShoot)<0.1f)
                _characterBehavior.BehaviorState.FiringDirection=3;

        bool _facingRight = transform.localScale.x > 0;
        float horizontalDirection=_facingRight?1f:-1f;

        direction = Quaternion.Euler(0,0,-angle) * direction;

        _weapon.GunRotationCenter.transform.rotation=
                Quaternion.Euler (new Vector3(0, 0, -angle*horizontalDirection+90f));

        var projectile = (Projectile)Instantiate
                (_weapon.Projectile,_weapon.ProjectileFireLocation.position,
                        _weapon.ProjectileFireLocation.rotation);
        projectile.Initialize(gameObject,direction,_controller.Speed);

        if (_weapon.GunShootFx!=null)
                SoundManager.Instance.PlaySound(_weapon.GunShootFx,transform.position);
```

```
        }

        private Vector3 RotatePointAroundPivot(Vector3 point, Vector3 pivot, float angle)
        {
                angle = angle*(Mathf.PI/180f);
                var rotatedX = Mathf.Cos(angle) * (point.x - pivot.x) - Mathf.Sin(angle) * (point.y-pivot.y) + pivot.x;
                var rotatedY = Mathf.Sin(angle) * (point.x - pivot.x) + Mathf.Cos(angle) * (point.y - pivot.y) + pivot.y;
                return new Vector3(rotatedX,rotatedY,0);
        }

        public void SetHorizontalMove(float value)
        {
                _horizontalMove=value;
        }

        public void SetVerticalMove(float value)
        {
                _verticalMove=value;
        }

        public void Flip()
        {
                if (_weapon.GunShells != null)
                        _weapon.GunShells.transform.eulerAngles =
                         new Vector3 (_weapon.GunShells.transform.eulerAngles.x,
                                 _weapon.GunShells.transform.eulerAngles.y +
                                 180, _weapon.GunShells.transform.eulerAngles.z);
                if (_weapon.GunFlames != null)
                        _weapon.GunFlames.transform.eulerAngles =
                                 new Vector3 (_weapon.GunFlames.transform.eulerAngles.x,
                                 _weapon.GunFlames.transform.eulerAngles.y +
                                 180, _weapon.GunFlames.transform.eulerAngles.z);
        }
}
```

10. CharacterJetpack.cs

```
using UnityEngine;
using System.Collections;

public class CharacterJetpack : MonoBehaviour
{

        public ParticleSystem Jetpack;

        public float JetpackForce = 2.5f;
        public bool JetpackUnlimited = false;
        public float JetpackFuelDuration = 5f;
        public float JetpackRefuelCooldown=1f;

        private CharacterBehavior _characterBehavior;
        private CorgiController _controller;

        void Start ()
        {
                _characterBehavior = GetComponent<CharacterBehavior>();
                _controller = GetComponent<CorgiController>();

                if (Jetpack!=null)
                {
                        Jetpack.enableEmission=false;
                        GUIManager.Instance.SetJetpackBar (!JetpackUnlimited);
                        _characterBehavior.BehaviorState.JetpackFuelDurationLeft = JetpackFuelDuration;

                }
        }
```

```
public void JetpackStart()
{
	if ((!_characterBehavior.Permissions.JetpackEnabled)||
		(!_characterBehavior.BehaviorState.CanJetpack)||
					(_characterBehavior.BehaviorState.IsDead))
		return;

	if (!_characterBehavior.BehaviorState.CanMoveFreely)
		return;

	if ((!JetpackUnlimited) && (_characterBehavior.BehaviorState.JetpackFuelDurationLeft <= 0f))
	{
		JetpackStop();
		_characterBehavior.BehaviorState.CanJetpack=false;
		return;
	}


	_controller.SetVerticalForce(JetpackForce);
	_characterBehavior.BehaviorState.Jetpacking=true;
	_characterBehavior.BehaviorState.CanMelee=false;
	_characterBehavior.BehaviorState.CanJump=false;
	Jetpack.enableEmission=true;
	if (!JetpackUnlimited)
	{
		StartCoroutine (JetpackFuelBurn ());

	}
}

public void JetpackStop()
{
	if (Jetpack==null)
		return;
	_characterBehavior.BehaviorState.Jetpacking=false;
	_characterBehavior.BehaviorState.CanMelee=true;
	Jetpack.enableEmission=false;
	_characterBehavior.BehaviorState.CanJump=true;

	if (!JetpackUnlimited)
		StartCoroutine (JetpackRefuel());
}


private IEnumerator JetpackFuelBurn()
{
	float timer=_characterBehavior.BehaviorState.JetpackFuelDurationLeft;
	while ((timer > 0) && (_characterBehavior.BehaviorState.Jetpacking))
	{
		timer -= Time.deltaTime;
		_characterBehavior.BehaviorState.JetpackFuelDurationLeft=timer;
		yield return 0;
	}
}

private IEnumerator JetpackRefuel()
{
	yield return new WaitForSeconds (JetpackRefuelCooldown);
	float timer=_characterBehavior.BehaviorState.JetpackFuelDurationLeft;
	while ((timer < JetpackFuelDuration) && (!_characterBehavior.BehaviorState.Jetpacking))
	{
		timer += Time.deltaTime/2;
		_characterBehavior.BehaviorState.JetpackFuelDurationLeft=timer;
		if ((!_characterBehavior.BehaviorState.CanJetpack) && (timer > 1f))
			_characterBehavior.BehaviorState.CanJetpack=true;
		yield return 0;
	}
}
```

```
}
```

11. InputManager.cs

```
using UnityEngine;
using System.Collections;
using UnitySampleAssets.CrossPlatformInput;

public class InputManager : PersistentSingleton<InputManager>
{

        public bool InDialogueZone;
        private static CharacterBehavior _player;
        private static CorgiController _controller;

        void Start()
        {
                InDialogueZone = false;
                if (GameManager.Instance.Player!=null)
                {
                        _player = GameManager.Instance.Player;
                        if (GameManager.Instance.Player.GetComponent<CorgiController>() != null)
                        {
                                _controller = GameManager.Instance.Player.
                                                        GetComponent<CorgiController>();
                        }
                }
        }

        void Update()
        {

                if (_player == null)
                {
                        if (GameManager.Instance.Player!=null)
                        {
                                if (GameManager.Instance.Player.GetComponent<CharacterBehavior>
                                    () != null)
                                        _player = GameManager.Instance.Player;
                                        _controller = GameManager.Instance.Player.
                                                        GetComponent<CorgiController>();
                        }
                        else
                                return;
                }

                if ( CrossPlatformInputManager.GetButtonDown("Pause") )
                        GameManager.Instance.Pause();

                if (GameManager.Instance.Paused)
                        return;

                if ((_player.BehaviorState.InDialogueZone)
                   &&(_player.BehaviorState.CurrentDialogueZone!=null)
                   &&(!_player.BehaviorState.IsDead)
                   &&(_controller.State.IsGrounded)
                   &&(!_player.BehaviorState.Dashing))
                {
                        if (CrossPlatformInputManager.GetButtonDown ("Jump"))
                        {
                                _player.BehaviorState.CurrentDialogueZone.StartDialogue();
                        }
                }

                if (!GameManager.Instance.CanMove)
                        return;

                _player.SetHorizontalMove(CrossPlatformInputManager.GetAxis ("Horizontal"));
                _player.SetVerticalMove(CrossPlatformInputManager.GetAxis ("Vertical"));
```

```
                if ((CrossPlatformInputManager.GetButtonDown("Run")||
                                CrossPlatformInputManager.GetButton("Run")) )
                        _player.RunStart();

                if (CrossPlatformInputManager.GetButtonUp("Run"))
                        _player.RunStop();


                if (CrossPlatformInputManager.GetButtonDown ("Jump"))
                {
                        if (!_player.BehaviorState.InDialogueZone)
                        {
                                _player.JumpStart ();
                        }
                }

                if (CrossPlatformInputManager.GetButtonUp("Jump"))
                {
                        _player.JumpStop();
                }

                if ( CrossPlatformInputManager.GetButtonDown("Dash") )
                        _player.Dash();


                if (_player.GetComponent<CharacterMelee>() != null)
                {
                        if ( CrossPlatformInputManager.GetButtonDown("Melee")  )
                                _player.GetComponent<CharacterMelee>().Melee();
                }

                if (_player.GetComponent<CharacterShoot>() != null)
                {
                        _player.GetComponent<CharacterShoot>().
                                SetHorizontalMove(CrossPlatformInputManager.GetAxis ("Horizontal"));
                        _player.GetComponent<CharacterShoot>().
                                SetVerticalMove(CrossPlatformInputManager.GetAxis ("Vertical"));

                        if (CrossPlatformInputManager.GetButtonDown("Fire"))
                                _player.GetComponent<CharacterShoot>().ShootOnce();


                        if (CrossPlatformInputManager.GetButton("Fire"))
                                _player.GetComponent<CharacterShoot>().ShootStart();

                        if (CrossPlatformInputManager.GetButtonUp("Fire"))
                                _player.GetComponent<CharacterShoot>().ShootStop();

                }

                if (_player.GetComponent<CharacterJetpack>()!=null)
                {
                        if ((CrossPlatformInputManager.GetButtonDown("Jetpack")
                                        ||CrossPlatformInputManager.GetButton("Jetpack")) )
                                _player.GetComponent<CharacterJetpack>().JetpackStart();

                        if (CrossPlatformInputManager.GetButtonUp("Jetpack"))
                                _player.GetComponent<CharacterJetpack>().JetpackStop();
                }
        }
}
```

12. GuiManager.cs

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

public class GUIManager : MonoBehaviour
{
	public GameObject HUD;
	public GameObject PauseScreen;
	public GameObject TimeSplash;
	public GameObject Buttons;
	public GameObject Pad;
	public Text PointsText;
	public Text LevelText;
	public Image Fader;
	public GameObject JetPackBar;

	private static GUIManager _instance;

	public static GUIManager Instance
	{
		get
		{
			if(_instance == null)
				_instance = GameObject.FindObjectOfType<GUIManager>();
			return _instance;
		}
	}

	public void Start()
	{
		RefreshPoints();
	}

	public void SetHUDActive(bool state)
	{
		HUD.SetActive(state);
		PointsText.enabled=state;
		LevelText.enabled=state;
	}

	public void SetMobileControlsActive(bool state)
	{
		Pad.SetActive(state);
		Buttons.SetActive(state);
	}

	public void SetPause(bool state)
	{
		PauseScreen.SetActive(state);
	}

	public void SetJetpackBar(bool state)
	{
		JetPackBar.SetActive(state);
	}

	{
		TimeSplash.SetActive(state);
	}

	public void RefreshPoints()
	{
		PointsText.text = "$" + GameManager.Instance.Points.ToString("000000");
	 }
```

```
	public void SetLevelName(string name)
	{
		LevelText.text=name;
	}

	public void FaderOn(bool state,float duration)
	{
		Fader.gameObject.SetActive(true);
		if (state)
			StartCoroutine(CorgiTools.FadeImage(Fader,duration, new Color(0,0,0,1f)));
		else
			StartCoroutine(CorgiTools.FadeImage(Fader,duration,new Color(0,0,0,0f)));
	}


}
```

13. CorgiSaves.cs

```
using UnityEngine;
using System.Collections;
using System.IO;

public class CorgiSaves : MonoBehaviour {

	public static CorgiSaves Instance;
	private bool saves = false;
	private bool tutorial = false;

	public static bool lvl1 = false;
	public static int maxgainlv1 = 20;
	public static bool lvl2 = false;
	public static int maxgainlv2 = 20;
	public static bool lvl3 = false;
	public static int maxgainlv3 = 30;
	public static bool lvl4 = false;
	public static int maxgainlv4 = 25;
	public static bool lvl5 = false;
	public static int maxgainlv5 = 25;

	void Awake()
	{
	 Instance = this;
	}

	void Update()
	{
		if(PlayerPrefs.GetInt("statesaves") == 1){  saves = true;   } else {  saves = false;  }
		if(PlayerPrefs.GetInt("statetutorial") > 0){  tutorial = true;   } else {  tutorial = false;  }
		LoadGame();
	}


	public int gaincoinlevel1
	{
		get { return PlayerPrefs.GetInt("getcoinlvl1"); }
		set { PlayerPrefs.SetInt("getcoinlvl1",value); }
	}

	public int gaincoinlevel2
	{
		get { return PlayerPrefs.GetInt("getcoinlvl2"); }
		set {  PlayerPrefs.SetInt("getcoinlvl2",value); }
	}
```

```
	public int gaincoinlevel3
	{
		get { return PlayerPrefs.GetInt("getcoinlvl3"); }
		set {  PlayerPrefs.SetInt("getcoinlvl3",value); }
	}

	public int gaincoinlevel4
	{
		get { return PlayerPrefs.GetInt("getcoinlvl4"); }
		set {  PlayerPrefs.SetInt("getcoinlvl4",value); }
	}

	public int gaincoinlevel5
	{
		get { return PlayerPrefs.GetInt("getcoinlvl5"); }
		set {  PlayerPrefs.SetInt("getcoinlvl5",value); }
	}

	public int scores
	{
		get { return PlayerPrefs.GetInt("Scores"); }
		set {  PlayerPrefs.SetInt("Scores",value); }
	}

	public void SaveGame()
	{
		PlayerPrefs.SetInt("getcoinlvl1",gaincoinlevel1);
		if(lvl1 == true) {PlayerPrefs.SetInt("lvl1",1);}
		 PlayerPrefs.SetInt("getcoinlvl2",gaincoinlevel2);
		if (lvl2 == true) { PlayerPrefs.SetInt("lvl2", 1); }
		PlayerPrefs.SetInt("getcoinlvl3",gaincoinlevel3);
		if (lvl3 == true) { PlayerPrefs.SetInt("lvl3", 1); }
		PlayerPrefs.SetInt("getcoinlvl4",gaincoinlevel4);
		if (lvl4 == true) { PlayerPrefs.SetInt("lvl4", 1); }
		PlayerPrefs.SetInt("getcoinlvl5",gaincoinlevel5);
		if (lvl5 == true) { PlayerPrefs.SetInt("lvl5", 1); }
		PlayerPrefs.SetInt("Scores",scores);
		if(saves == true){PlayerPrefs.SetInt("statesaves",1);}
		if(tutorial == true){PlayerPrefs.SetInt("statetutorial",2);}
		PlayerPrefs.Save();
		LoadGame();
	}

	public void LoadGame()
	{
		gaincoinlevel1 = PlayerPrefs.GetInt("getcoinlvl1");
		if(gaincoinlevel1 > 0 || tutorial == true){ lvl1 = true; }
		gaincoinlevel2 = PlayerPrefs.GetInt("getcoinlvl2");
		if(PlayerPrefs.GetInt("lvl2") == 1){ lvl2 = true; }
		gaincoinlevel3 = PlayerPrefs.GetInt("getcoinlvl3");
		if(PlayerPrefs.GetInt("lvl3") == 1) { lvl3 = true; }
		gaincoinlevel4 = PlayerPrefs.GetInt("getcoinlvl4");
		if(PlayerPrefs.GetInt("lvl4") == 1) { lvl4 = true; }
		gaincoinlevel5 = PlayerPrefs.GetInt("getcoinlvl5");
		if(PlayerPrefs.GetInt("lvl5") == 1) { lvl5 = true; }

		scores = PlayerPrefs.GetInt("Scores");
		if(PlayerPrefs.GetInt("statesaves") == 1){ saves = true; }
		if(PlayerPrefs.GetInt("statetutorial") > 0){ tutorial = true; }
	}
}
```

14. GameManager.cs

```
using UnityEngine;
using System.Collections;

public class GameManager : PersistentSingleton<GameManager>
{
  public int Points
  {
    get { return PlayerPrefs.GetInt("Scores"); }
    set { PlayerPrefs.SetInt("Scores", value); }
  }
  public float TimeScale { get; private set; }
        public bool Paused { get; set; }
        public bool CanMove=true;
        public CharacterBehavior Player { get; set; }

        private float _savedTimeScale;

        public void Reset()
        {
                TimeScale = 1f;
                Paused = false;
                CanMove=false;
                GUIManager.Instance.RefreshPoints();
        }

        public void AddPoints(int pointsToAdd)
        {
                Points += pointsToAdd;
                GUIManager.Instance.RefreshPoints();
        }

        public void SetPoints(int points)
        {
                Points = points;
                GUIManager.Instance.RefreshPoints ();
        }

        public void SetTimeScale(float newTimeScale)
        {
                _savedTimeScale = Time.timeScale;
                Time.timeScale = newTimeScale;
        }

        public void ResetTimeScale()
        {
                Time.timeScale = _savedTimeScale;
        }

        public void Pause()
        {
                if (Time.timeScale>0.0f)
                {
                        Instance.SetTimeScale(0.0f);
                        Instance.Paused=true;
                        GUIManager.Instance.SetPause(true);
                }
                else
                {
                        Instance.ResetTimeScale();
                        Instance.Paused=false;
                        GUIManager.Instance.SetPause(false);
                }
        }
```

```
	public void FreezeCharacter()
	{
		Player.SetHorizontalMove(0);
		Player.SetVerticalMove(0);
		Instance.CanMove=false;
	}
}
```

15. StartMenu.cs

```
using UnityEngine;
using System.Collections;

public class StartMenu : MonoBehaviour {

	private float delay = 3f;
	public GameObject ContinueText;
	public GameObject Objectmenu;
	public GameObject Objecttitle;
	public GameObject Objectexitgame;
	public GameObject Objectconfirm;
	public GameObject LoadingText;
	public GameObject aboutpanel;
	 public GameObject secret;

	void Update()
	{
		if (PlayerPrefs.GetInt("statesaves") == 1){ ContinueText.SetActive(true); } else {
			ContinueText.SetActive(false); }
		if (PlayerPrefs.GetInt("Scores") >= 10000 && CorgiSaves.Instance.gaincoinlevel1 == 20
		 && CorgiSaves.Instance.gaincoinlevel2 == 20 && CorgiSaves.Instance.gaincoinlevel3 == 30
		&& CorgiSaves.Instance.gaincoinlevel4 == 25 && CorgiSaves.Instance.gaincoinlevel5 == 25 )
		{ secret.SetActive(true); }
	}

	public void FirstTime()
	{
		if(PlayerPrefs.GetInt("statesaves") == 1)
		{
			Objecttitle.SetActive(false);
			Objectmenu.SetActive(false);
			Objectconfirm.SetActive(true);
		}
		else
		{
			firststart();
		}
	}

	void firststart()
	{
		LoadingText.SetActive(true);
		Objecttitle.SetActive(true);
		Objectconfirm.SetActive(false);
		Objectmenu.SetActive(false);
		PlayerPrefs.DeleteAll();
		StartCoroutine(LoadLevel(2));
	}

	IEnumerator LoadLevel(int level)
	{
		yield return new WaitForSeconds(delay);
		Application.LoadLevel(level);
	}
```

```
	public void returnmenu()
	{
		Objecttitle.SetActive(true);
		Objectmenu.SetActive(true);
		Objectconfirm.SetActive(false);
		Objectexitgame.SetActive(false);
	}

	public void Continue()
	{
		if(PlayerPrefs.GetInt("statetutorial") > 1)
		{
		    LoadingText.SetActive(true);
		    Objectmenu.SetActive(false);
		    StartCoroutine(LoadLevel(1));
		}
		else
		{
		    LoadingText.SetActive(true);
		    Objectmenu.SetActive(false);
		    StartCoroutine(LoadLevel(2));
		}
	}

	public void ConfirmExit()
	{
		Objectexitgame.SetActive(true);
		Objectmenu.SetActive(false);
	}

	public void AboutPanelactive()
	{
		aboutpanel.SetActive(true);
		Objectmenu.SetActive(false);
		Objecttitle.SetActive(false);
	}

	public void AboutPaneldeactive()
	{
		aboutpanel.SetActive(false);
		Objectmenu.SetActive(true);
		Objecttitle.SetActive(true);
	}

	public void addingsecretstage()
	{
	  PlayerPrefs.SetInt("Scores", 10000);
	  CorgiSaves.lvl1 = true;
	  CorgiSaves.Instance.gaincoinlevel1 = 20;
	  CorgiSaves.lvl2 = true;
	  CorgiSaves.Instance.gaincoinlevel2 = 20;
	  CorgiSaves.lvl3 = true;
	  CorgiSaves.Instance.gaincoinlevel3 = 30;
	  CorgiSaves.lvl4 = true;
	  CorgiSaves.Instance.gaincoinlevel4 = 25;
	  CorgiSaves.lvl5 = true;
	  CorgiSaves.Instance.gaincoinlevel5 = 25;
	  PlayerPrefs.SetInt("statesaves",1);
	  PlayerPrefs.SetInt("statetutorial",2);
	  secret.SetActive(true);
	}

	public void ExitGame()
	{
		CorgiSaves.Instance.SaveGame();
		Application.Quit();
	}
}
```

16. InGameMenu.cs

```
using UnityEngine;
using System.Collections;

public class InGameMenu : MonoBehaviour {

	public InGameMenu Instance;
	public GameObject pausebtn;
	public GameObject pausetext;
	public GameObject pausemenu;
	public GameObject confirmpnl;
	public GameObject troublefader;
	public string LevelName;
	private int countcoins;
	private string names;

	void Awake()
	{
	Instance = this;
	}

	public void PauseGame()
	{
		GameManager.Instance.SetTimeScale(0);
		GameManager.Instance.Paused = true;
		GUIManager.Instance.SetPause(true);

		pausetext.SetActive(true);
		pausemenu.SetActive(true);
		pausebtn.SetActive(false);
		troublefader.SetActive(false);
	}

	public void ResumeGame()
	{
		GameManager.Instance.ResetTimeScale();
		GameManager.Instance.Paused = false;
		GUIManager.Instance.SetPause(false);

		pausetext.SetActive(false);
		pausemenu.SetActive(false);
		pausebtn.SetActive(true);
		troublefader.SetActive(true);
	}

	public void ConfirmtoTitle()
	{
		confirmpnl.SetActive(true);
		pausemenu.SetActive(false);
	}

	public void BacktoMenu()
	{
		confirmpnl.SetActive(false);
		pausemenu.SetActive(true);
	}

	public void RestartGame()
	{
	  Application.LoadLevel(Application.loadedLevelName);
	  pausetext.SetActive(false);
	  pausemenu.SetActive(false);
	}

	public void BacktoTitle()
	{
		if(PlayerPrefs.GetInt("statesaves") == 0)
		{
```

```
                                PlayerPrefs.SetInt("statesaves",1);
                                PlayerPrefs.SetInt("statetutorial",1);
                        }

                if (LevelName == "LevelSelect")
                {
                    names = CheckPoint.Instance.levelname.ToString();

                    switch (names)
                    {
                        case "Level1": countcoins = Coin.Instance.count; if (countcoins > CorgiSaves.maxgainlv1) {
                                CorgiSaves.Instance.gaincoinlevel1 = 20;
                                countcoins = 0; } else { CorgiSaves.Instance.gaincoinlevel1 = countcoins;
                                countcoins = 0; }
                                 break;
                        case "Level2": countcoins = Coin.Instance.count; if (countcoins > CorgiSaves.maxgainlv2) {
                                CorgiSaves.Instance.gaincoinlevel2 = 20; countcoins = 0; } else {
                                CorgiSaves.Instance.gaincoinlevel2 = countcoins; countcoins = 0; }
                                break;
                        case "Level3": countcoins = Coin.Instance.count; if (countcoins > CorgiSaves.maxgainlv3) {
                                CorgiSaves.Instance.gaincoinlevel3 = 30; countcoins = 0; } else {
                                CorgiSaves.Instance.gaincoinlevel3 = countcoins; countcoins = 0; }
                                break;
                        case "Level4": countcoins = Coin.Instance.count; if (countcoins > CorgiSaves.maxgainlv4) {
                                CorgiSaves.Instance.gaincoinlevel4 = 25; countcoins = 0; } else {
                                CorgiSaves.Instance.gaincoinlevel4 = countcoins; countcoins = 0; }
                                break;
                        case "Level5": countcoins = Coin.Instance.count; if (countcoins > CorgiSaves.maxgainlv5) {
                                CorgiSaves.Instance.gaincoinlevel5 = 25; countcoins = 0; } else {
                                CorgiSaves.Instance.gaincoinlevel5 = countcoins; countcoins = 0; }
                                break;
                    }
                }
                PlayerPrefs.SetInt("Scores", GameManager.Instance.Points);

                GameManager.Instance.ResetTimeScale();
                        Application.LoadLevel(1);
                }
            }
```

17. LevelSelector.cs

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

public class LevelSelector : MonoBehaviour {

    public Text gainlvl1;
    public Text maxlvl1;
    public Text gainlvl2;
    public Text maxlvl2;
    public Text gainlvl3;
    public Text maxlvl3;
    public Text gainlvl4;
    public Text maxlvl4;
    public Text gainlvl5;
    public Text maxlvl5;
    public Text highscores;
    public GameObject buttonlv1;
    public GameObject buttonlv2;
    public GameObject buttonlv3;
    public GameObject buttonlv4;
    public GameObject buttonlv5;
    public GameObject backtomenu;
```

```
void Start()
{
   //locked
        if (CorgiSaves.lvl1 == false) { buttonlv1.GetComponent<Button>().interactable = false; } else {
        buttonlv1.GetComponent<Button>().interactable = true; }
        if (CorgiSaves.lvl2 == false) { buttonlv2.GetComponent<Button>().interactable = false; } else {
        buttonlv2.GetComponent<Button>().interactable = true; }
        if (CorgiSaves.lvl3 == false) { buttonlv3.GetComponent<Button>().interactable = false; } else {
        buttonlv3.GetComponent<Button>().interactable = true; }
        if (CorgiSaves.lvl4 == false) { buttonlv4.GetComponent<Button>().interactable = false; } else {
        buttonlv4.GetComponent<Button>().interactable = true; }
        if (CorgiSaves.lvl5 == false) { buttonlv5.GetComponent<Button>().interactable = false; } else {
        buttonlv5.GetComponent<Button>().interactable = true; }

   //gain on level
   gainlvl1.text = CorgiSaves.Instance.gaincoinlevel1.ToString();
   gainlvl2.text = CorgiSaves.Instance.gaincoinlevel2.ToString();
   gainlvl3.text = CorgiSaves.Instance.gaincoinlevel3.ToString();
   gainlvl4.text = CorgiSaves.Instance.gaincoinlevel4.ToString();
   gainlvl5.text = CorgiSaves.Instance.gaincoinlevel5.ToString();
   highscores.text = CorgiSaves.Instance.scores.ToString();

   //max gain perlevel
   maxlvl1.text = "/" + CorgiSaves.maxgainlv1.ToString();
   maxlvl2.text = "/" + CorgiSaves.maxgainlv2.ToString();
   maxlvl3.text = "/" + CorgiSaves.maxgainlv3.ToString();
   maxlvl4.text = "/" + CorgiSaves.maxgainlv4.ToString();
   maxlvl5.text = "/" + CorgiSaves.maxgainlv5.ToString();

}

void Update()
{
   highscores.text = PlayerPrefs.GetInt("Scores").ToString();

   if (CorgiSaves.lvl1 == false) { buttonlv1.GetComponent<Button>().interactable = false; } else {
        buttonlv1.GetComponent<Button>().interactable = true; }
   if (CorgiSaves.lvl2 == false) { buttonlv2.GetComponent<Button>().interactable = false; } else {
        buttonlv2.GetComponent<Button>().interactable = true; }
   if (CorgiSaves.lvl3 == false) { buttonlv3.GetComponent<Button>().interactable = false; } else {
        buttonlv3.GetComponent<Button>().interactable = true; }
   if (CorgiSaves.lvl4 == false) { buttonlv4.GetComponent<Button>().interactable = false; } else {
        buttonlv4.GetComponent<Button>().interactable = true; }
   if (CorgiSaves.lvl5 == false) { buttonlv5.GetComponent<Button>().interactable = false; } else {
        buttonlv5.GetComponent<Button>().interactable = true; }

   CorgiSaves.Instance.SaveGame();

        public void gotolevel(string levelname)
        {
                Application.LoadLevel(levelname);
        }

        public void backtomainmenu()
        {
                Application.LoadLevel(0);
        }

}
```