BAB II

TINJAUAN PUSTAKA

2.1 Tinjauan Teori Umum

2.1.1 Sistem Informasi

Dapat dikatakan sistem ialah perkumpulan dari bagian yang saling terhubungan dengan maksud supaya dapat mencapai suatu tujuan. Apabila dalam suatu sistem memiliki bagian yang tidak dapat memberikan manfaat, dengan begitu bagian itu sudah pasti bukanlah bagian dari suatu sistem. Informasi termasuk pemrosesan suatu data yang dapat meningkatkan wawasan pada pengguna infrmasi itu. Dan informasi juga merupakan suatu data yang telah diolah menjadi suatu bentuk yang mempunyai makna dan manfaat untuk orang yang menerimanya pada saat melakukan pengambilan keputusan pada masa sekarang ini maupun masa yang akan datang (Aventinus, 2020).

Kemudian yang dimaksud dari sistem informasi yaitu penggabungan dari empat bagian utama. Dari keempat bagian utama itu diantaranya termasuk *hardware, software*, SDM yang terlatih dan juga infrastruktur (Ultariani et al., 2020).

2.1.2 Perancangan dan Pembangunan Perangkat Lunak

Suatu *software* dapat dikatakan baik dan efisien, apabila *software* tersebut bisa mempersiapkan seluruh kebutuhan yang dibutuhkan dalam memudahkan pengguna pada saat menyelesaikan permasalahan, atau berdasarkan pada perjanjian yang terdapat pada kontrak (Setyowardani et al., n.d.).

Sebuah *software* dapat dikatakan berkualitas apabila *software* tersebut bisa memuaskan sebagian besar bagi orang yang menggunakannya. Suatu produk dapat dikatakan memiliki kualitas apabila produk tersebut bisa memberikan kepastian terhadap produknya memenuhi mutu yang telah ditentukan. Atau dapat dikatakan menguji kualitas pada perangkat lunak termasuk bagian dari jaminan pada kualitas *software* yang menjelaskan inti dari desain, pengkodean dan spesifikasi (Andriyani et al., 2021).

Kemudian pegujian ini memiliki tujuan dalam memperlihatkan kesamaan pada fungsi *software* pada spesifikasinya. Sebuah *software* dapat dikatakan gagal, apabila *software* tersebut tidak memenuhi dari spesifikasi yang telah ditentukan. Sebelum sebuah *software* digunakan, maka perlu dilakukan uji coba supaya menjumpai kesalahan-kesalahan yang tidak diinginkan. Pengujian ini merupakan tahapan yang paling mahal dalam mengembangkan sebuah perangkat lunak supaya *software* tersebut bebas dari kesalahan umunya (Setiawan et al., 2020).

Model dalam mengembangkan *software* yang sering digunakan yaitu dengan model *waterfall*. Pada model ini memiliki sifat linear dimulai dari tahapan awal dalam merencanakan pngembangan sistem sampai dengan tahapan akhir yaitu tahapan pemeliharaan (Wahid, 2020). Tahapan-tahapan pada umumnya yaitu:

1. Analisis, dimana komunikasi diperlukan dalam mengembangkan sistem dengan tujuan supaya dapat mengerti *software* yang pengguna harapkan dan

pembatasan pada *software* itu. Proses analisis diperlukan keterlibatan pengguna untuk mendapatkan data yang dibutuhkan.

2. Desain, dimana desin sistem dibuat oleh pengembang supaya dapat membantu dalam menetaplam *hardware* dan kriteria dalam suatu sistem serta juga dalam membantu merancang *hardware* yang didalamnya terdapat gambaran tampilan aplikasi seperti tampilan aplikasi yang secara umum dan tampilan antarmuka.
3. Implementasi, dimana pertama kali pengembang memulai sistem dala program yang kecil atau biasanya disebut sebagai unit, yang terpadu pada tahapan berikutnya. Setiap unit diuji serta dikembangkan memiliki fungsionalitas pada *software*. Aktivitas yang termasuk pemograman, pembuatan aset seperti musik, suara, gambar dan aset lain yang nanti akan digabungkan bersamaan.
4. Pengujian, dimana pengujian pada sistem apakah sistem dengan sebagian atau sepenuhnya memenuhi syarat-syarat pada sistem, pengujian bisa di golongkan dalam unit testing, pengujian pada sistem aakn beraksi saat semua modul menyatu dan penerimaan dalam pengujian dalam melihat bagaimana keberhasilan suatu sistem dalam aplikasi bisa memenuhi kebutuhan bagi penggunanya, kemampuan aplikasi meminimalkan kesalahan seperti (*bug dan glitch*), serta kemampuan aplikasi dalam menjalankan fungsinya. Jika ditemukan kesalahan, maka akan dilakukan peerbaikan dan pengujian lagi sampai masalah tersebut selesai teratasi.

5. Pemeliharaan, yang merupakan metode yang mana *software* yang telah selesai dirancang, kemudian dijalankan serta dapat diperbaharui dengan fitur tambahan. Hal ini merupakan tahapan akhir dalam metode *waterfall*. *Software* yang telah selesai dirancang, akan dijalankan dan dilakukan pemeliharaan. Di dalam pemeliharaan ini sudah termasuk perbaikan atas kesalahan yang tidak didapati di langkah sebelumnya.

Model *waterfall* merupakan model *System Development Life Cycle* (SDLC) yang paling sederhana. Dalam model ini sangat serasi dalam mengembangkan *software* pada spesifikasi yang tidak beraturan. Pada model ini juga memiliki beberapa kelemahan dan model ini juga sebagai dasar dari model yang lain pada saat memperbaiki pengembangan *software.* Model ini sangat mudah dipahami oleh pengguna dan memungkinkan terjadinya perubahan pada saat mengembangkan perangkat *software*.

Seiring berkembangnya dalam dunia industri *game*, munculnya *game* lebih berkualitas dan menarik dari segi visualisasi ataupun dari segi cerita juga semakin banyak. Supaya pada era globalisasi ini membuat setiap warga negara mempunyai perasaan cinta terhadap tanah air pada seluruh budaya serta perkembangan pada teknologi informasi termasuk *game* yang disebarluaskan pada kehidupan ini. Sekarang ini *game* merupakan perangkat lunak yang sedang mengalami perkembangan. Berbagai macam *game* yang memiliki tampilan yang lebih menarih dan dijadikan sebagai alat *refreshing* (Armanda & Rizqi, 2020).

Untuk saat ini, *video game* belum hendak memperlihatkan pada dunia publik. Pengembang *video game* mengambil keputusan seperti ini supaya bisa memberikan kejutan ke calon pembeli yang memiliki minat untuk melakukan pembelian pada *video game* itu. Apabila keputusan itu diambil, dengan begitu pihak lain akan dijadikan sebagai calon pembeli. Pihak penguji biasanya disebut sebagai *beta tester*. Setelah *video game* selesai dikembangkan, bukan berarti *video game* itu langsung akan diterima oleh khalayak ramai. *Beta testing* atau yang disebut sebagai *eksternal tersting* dilakuakn pada saat melakukan pengujian pada *video game* dalam menemukan berbagai *bug* atau *error* atau permasalahan yang lain yang diperoleh dari pihak bermain. *Beta* berada pada luar *production cycle*, akan tetapi hasil dari *testing* mempunyai kemampuan yang membuat tim melakukan pengulangan pada *production cycle* (Prasetyo et al., 2021). Pihak penguji umumnya sudah terikat kontrak dengan pengembang sebagai tujuan untuk tidak membocorkan komponen yang ada yang terdapat pada *video game* yang sedang dirancang itu.

2.1.3 Aliran Sistem Informasi

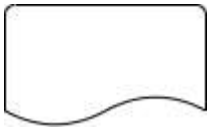
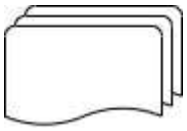
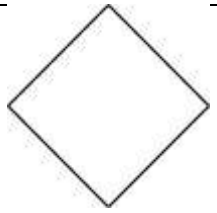
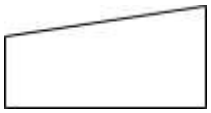
Supaya melihat suatu sistem yang ada dan mengetahui sistem informasi yang sedang berjalan itu, diperlukan memberikan penjelasan pada aliran sistem informasi mengenai elemen penting yang terdapat di luar ataupun di dalam sistem. Dari hal tersebut lebih memudahkan pemahaman tentang informasi yang sudah ditemukan dan ditekan pada sistem itu dengan sendirinya (Tukino & Arnomo, 2021).

Apabila sistem informasi tersebut tidak sebanding, dengan begitu perlu dilakukan perubahan pada olahan datanya sehingga dapat menghasilkan informasi dengan akurat dan cepat serta pengambilan keputusan menjadi lebih baik. Aliran sistem informasi dapat menjelaskan proses berjalannya pada sistem informasi (Yulisman, 2019). Hal ini serupa dengan *flowchart*, dimana aliran pada sistem informasi dibuat supaya memperlihatkan kepada pihak yang ikut serta pada proses berjalannya suatu sistem. Sistem yang baru saja dikembangkan juga dapat menggunakan aliran pada sistem informasi ini. Adapun simbol dari Aliran Sistem Informasi (ASI), diantaranya:

Tabel 2.1 Simbol-simbol yang akan digunakan dalam aliran sistem informasi

No	Simbol	Nama	Penjelasan
1		*Start/End*	Melambangkan alur awal ataupun akhir
2		*Process*	Melambangkan aktivitas yang bisa dilakukan pada sistem

Tabel 2.1 Lanjutan

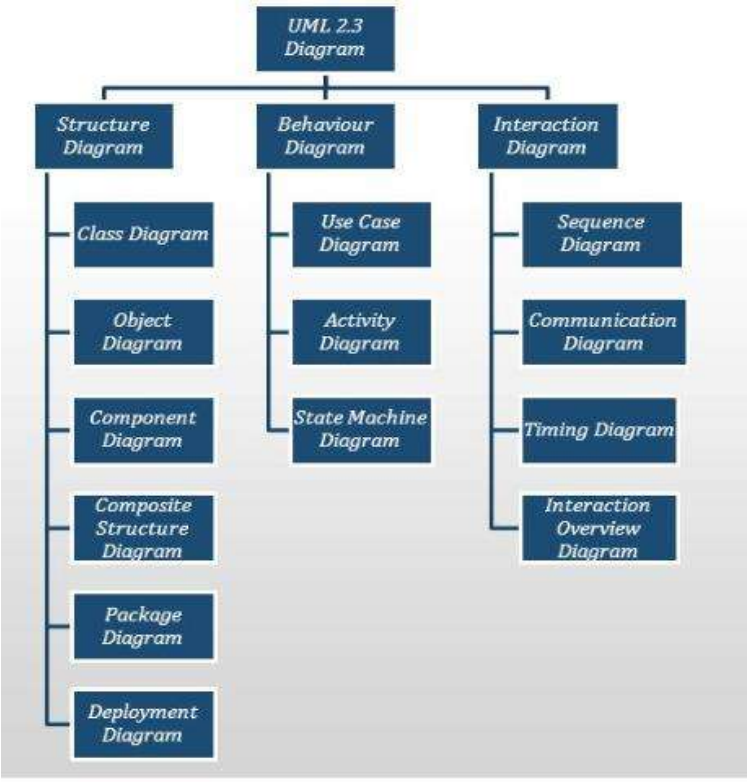
3		*Document*	Melambangkan keterlibatan dokumen terhadap sistem
4		*Multiple Document*	Melambangkan sejumlah dokumen yang memiliki keterlibatan terhadap sistem
5		*Decision*	Simbol yang memperlihatkan pada kejadian percabangan
6		*Input/Output*	Melambangkan data atau informasi yang masuk ataupun keluar
7		*Manual Input*	Memperlihatkan data atau informasi pada data yang sudah di *input* oleh pihak yang menggunakannya

2.1.4 *Unified Model Languange*

Pada dunia industri, bahasa yang banyak digunakan untuk membuat analisis, menjelaskan analisis, menggambarkan arsitektur dan desain dalam pemrograman

yang berfokus pada objek ialah UML. Yang dimaksud dengan UML yaitu suatu sarana pada bidang pengembangan sistem yang berfokus pada objek. Hal tersebut dikarenakan UML mempersiapkan bahasan pemodelan visal yang dapat memungkinkan pengembang sistem dalam membuat bentuk yang baku dan cetak baru dari visi mereka. UML juga memiliki fungsi sebagai jembatan pada saat melakukan komunikasi pada beberapa sudut pandang dari sistem (Gunawan et al., 2019).

UML memiliki tiga konsep abstraksi yang bisa dilakukan pada berbagai jenis diagram, diantaranya *use case diagram* yang biasanya digunakan dalam menggambatkan kelakukuan pada sistem yang akan dibuat, *activity diagram* yang biasanya digunakan dalam menggambarkan alur bekerja pada sistem, *sequence diagram* yang biasanya digunakan menggambarkan kerja yang terdapat pada objek, *class diagram* yang biasanya digunakan menggambarkan bagaimana suatu operasi dilaksanakan, dan yang terakhir *deployment diagram* yang biasanya digunakan menggambarkan elemen sistem yang digabungkan (Ependi, 2018). UML dibagi menjadi tiga kelompok, diantaranya: *structure diagram*, *behavior diagram*, dan *interaction diagram* yang setiap diagram bisa diperlihatkan pada gambar berikut ini.

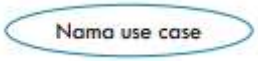


Gambar 2.1 Diagram pada UML

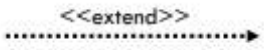
Sesuai pada gambar 2.1 merupakan langkah dalam merancang sebuah *video game*, digunakan diagram *use case, class, activity,* dan *sequence diagram.* Diagram-diagram itu digunakan sebagai perwakilan pada diagram yang lainnya.

1. Diagram *use case* termasuk model untuk menunjukkan *behavior* atau kelakukan pada sistem informasi yang akan dibuatnya (Syarif & Nugraha, 2020). *Use case* menerangkan sebuah hubungan yang terjadi antara sistem yang akan dibuat dengan aktor. *Use case* pada tugas akhir ini digunakan untuk mamberikan gambaran fungsi atau fungsional yang terdapat didalam sistam dan untuk mengetahui fungsi-fungsi tersebut dapat digunakan oleh siapa saja.

Tabel 2.2 Simbol yang terdapat pada *use case diagram*

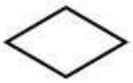
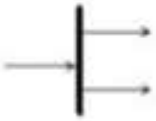
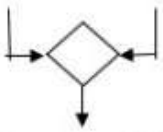
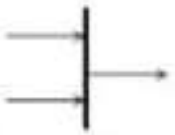
No	Simbol	Nama	Penjelasan
1		*Actor*	Melambangkan pihak yang mengharapkan *output* dari sistem dan juga dapat men-*input* data.
2		*Use Case*	Memperliharkan aktivitas yang bisa dilakukan oleh sistem.
3		*Include*	Melambangkan interaksi antar *use case* yang akan ikut terjadi.

Tabel 2.2 Lanjutan

4		*Extend*	Memperlihatkan relasi antar *use case* yang bisa terjadi tergantung dari campur tangan dari sistem atau *input.*
5		*Generalization*	Memperlihatkan relasi antar *use case* yang kemudian disederhanakan.
6		*Communication*	Memperlihatkan relasi antara *use case* dengan akhir yang bisa dilakukan pada aktor.

2. *Activity diagram* biasanya digunakan dalam menggambarkan berbagai jenis alur dari kegiatan pada sistem yang akan dikembangkan, kemudian bagaimana dimulainya alur itu, pengambilan keputusan yang mungkin terjadi serta bagaimana alur tersebut berakhir. *Activity diagram* menunjukkan aktivitas sistem berbentuk perkumpulan aksi, kemudian bagaimana aksi itu mulai dilakukan, dan keputusan yang kemungkinan terjadi sampai aksi itu berakhir (Asmasari & Informatika, 2020).

Tabel 2.3 Simbol-simbol yang terdapat pada *activity diagram*

No	Simbol	Nama	Penjelasan
1		*InitialState/Start*	Melambangkan status awal dari diagram
2		*Decision*	Melambangkan peristiwa pada keadaan pilihan
3		*Fork*	Melambangkan permulaan dari kegiatan yang secara bersamaan dilakukan
4		*Merge*	Melambangkan peristiwa akhir pada keadaan pilihan
5		*Join*	Melambangkan berbagai aktivitas terkhir yang pada sebelumnya telah berjalan dengan bersamaan

Tabel 2.3 Lanjutan

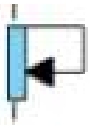
6		*FinalState/End*	Melambangkan berakhirnya diagram
7		*Action*	Melambangkan aksi
8		*Object*	Melambangkan objek

3. *Sequence diagram* biasanya digunakan pada saat memberikan penjelasan pada hubungan antara objek yang terdapat dalam sistem dengan memberikan keterangan penambahan urutan waktu (Maylawati et al., 2018). Masing-masing interaksi akan disusun sesuai dengan waktu interaksi tersebut terjadi, dan akan dimulai dari atas sampai dengan bawah. Diagram ini tidak akan menjelaskan tentang apa yang akan terjadi pada interaksi ini. Tujuan dari diagram ini yaitu untuk memberikan penjelasan secara berurutan proses terjadinya interaksi anta objek yang terdapat pada sistem.

Tabel 2.4 Simbol-simbol yang terdapat pada *sequence diagram*

No	Simbol	Nama	Penjelasan
1		*Actor*	Melambangkan pihak yang mengharapkan *output* dan akan melakukan *input* pada data.
2		*Object*	Melambangkan objek terdapat dalam sistem yang memiliki keterlibatan pada saat terjadinya interaksi.
3		*Destroy*	Melambangkan objek akan berhenti melakukan interaksinya.
4		*Lifeline*	Melambnagkan jangka waktu dalam keterlibatan aktor atau objek yang berlangsung selama interaksi.
5		*Activation*	Melambangkan aktor atau objek sedang berinteraksi ataupun sedang beroperasi.

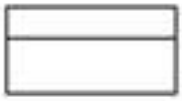
Tabel 2.4 Lanjutan

6		*Call message*	Melambangkan jenis operasi atau interaksi yang nantinya akan dikirim ke aktor atau objek lain.
7		*Return message*	Melambangkan jenis operasi atau interaksi yang nantinya akan dibalas ke aktor atau objek pengirim.
8		*Self message*	Melambangkan jenis operasi atau interaksi yang nantinya akan diterima dan dikirim ke *lifeline* aktor atau objek yang serupa.

4. *Class diagram* biasanya digunakan dalam memberikan penjelasan hubungan antar kelas. Setiap kelas memiliki bagian operasi dan atribut. Simbol relasi dapat digunakan dalam menghubungkan antar kelas. Diagram ini juga memudahkan para pembaca untuk memperoleh gambaran umum tentang

software yang akan dikembangkan (*UML Class Diagram Tutorial | Lucidchart*, n.d.).

Tabel 2.5 Simbol *class diagram*

No	Simbol	Nama	Penjelasan
1		*Class*	Melambangkan objek yang terdapat dalam sistem dan mempunyai kemampuan operasi dan atribut.
2		*Association*	Melambangkan hubungan antar kelas, tanpa memiliki ketergantungan yang khusus.
3		*Aggregation*	Melambangkan relasi yang khusus antar kelas, seperti *child class* termasuk pada elemen dari *parent class*. Akan tetapi *child class* juga bisa berdiri dengan sendirinya.

Tabel 2.5 Lanjutan

4		*Composition*	Melambangkan relasi yang khusus antar kelas, seperti bergantungan hidup terhadap *parent class*. *child class* akan menjadi tidak berlaku apabila *parent class* tidak ada.
5		*Inheritance*	Melambangkan relasi khusus antar kelas, seperti perlengkapan yang terdapat pada *parent class* dan juga yang terdapat dalam *child class*.
6		*Dependency*	Melambangkan relasi yang khusus antar kelas, seperti respon pada perubahan yang akan terjadi dalam *parent class*.

2.2 Teori Khusus

2.2.1 *Video Game*

Yang dimaksud dengan *video game* yaitu permainan yang bersifat elektronik dan yang melibatkan interaksi antar pengguna untuk mendapatkan umpan balik secara visual terhadap perangkat video. *Video game* berasal dari kata tradisional yaitu perangkat layar *raster*. Akan tetapi sampai sekarang ini digunakan sebagai sebutan *video game*, dan nama *video game* digunakan untuk menyebutkan permainan yang terdapat perangkat layar apapun. *Video game* yang khusus seperti *game arcade,* telah umum dan sering digunakan. *Video game* juga dikenal sebagai suatu bentuk seni (Acep, 2018).

Game dalam kamus bahasa indonesia memiliki arti yaitu permainan. G*ame* adalah suatu aktifitas baik itu terstruktur maupun semi terstruktur yang bertujuan sebagai sarana hiburan dan kadang untuk Pendidikan (Pradana & Nita, 2019).

Gambar 2.2 Contoh video game genre Battle Royale

Misalnya pada jenis *game* yang banyak dikenal oleh masyarakat, terutama untuk pengguna perangkat *smartphone* ialah *battle royale*, dengan *sub-genre* tembak-tembakan (*shooter*). Hal penting yang didapatkan dari jenis *game* ini yaitu memiliki satu sesi dengan jumlah pemain lebih dari 50 yang saling bertarung antara yang saru dengan yang lainnya, dan akan tersisa satu pemain yang masih bertahan hidup atau dikatakan sebagai pemenang dari *game* tersebut (GyuHyeok Choi, 2018). *Game* yang memiliki *sub-genre shooter*, akan menggunakan perangkap dan senjata api sebagai sarana dalam membuat musuh kalah (Karavolos et al., 2018). Contoh lain dari *game* jenis ini yaitu *Free Fire, Rules of Survival, Fortnite, Apex Legend, PlayerUnknown's BattleGround* dan game lainnya.

Gambar 2.3 Contoh *video game genre* simulasi

Pada permainan yang bersifat tradisional atau *game* yang telah ada pada dunia nyata juga bisa dikembangkan menjadi bentuk versi digital. Misalnya seperti catur, ular tangga, monopoli dan permaian lainnya. Selain itu pekerjaan dan kegiatan juga bisa dijadikan sebagai ide dalam mengembangkan sebuah *video*

game atau biasanya disebut sebagai *simulation*. Misalnya jenis *video game* simulasi ialah *Overcooked, Monopoly, Chess, The Sims,* dan lain sebagainya.

2.2.2 *Game Engine*

Dalam menjalankan *video game* yang pada umumnya atau aplikasi simulasi tiga dimensi akan dijalankan menggunakan aplikasi yang khusus. Dalam mengolah objek yang berbentuk tiga dimensi, biasanya akan menggunakan aplikasi yang khusus seperti Autodesk Maya atau AutoCAS. Selain itu, sebagai tempat *video game* tersebut dijalankan juga membutuhkan aplikasi yang khusus. Sebutan yang biasanya digunakan yakni *game engine* (Fachroni et al., 2018). *Game Engine* ini dibutukan supaya memudahkan dalam melakukan pengembangan, produksi dan pengujian pada *video game*. Apabila pengembang atau *developer* dari pengembang *video game* mempunyai kompetensi, bisa juga dengan membuat *game engine* dengan sendirinya. Misalnya seperti Godot Engine, Cry Engine, Unreal Engine, Unity, dan berbagai jenis *software* lainnya (*Features - Unreal Engine*, n.d.; *Godot Engine - Free and open source 2D and 3D game engine*, n.d.; *Solutions | Unity*, n.d.).

Gambar 2.4 Contoh *game engine* untuk pengembangan *video game*

2.2.3 Genre *Platformer*

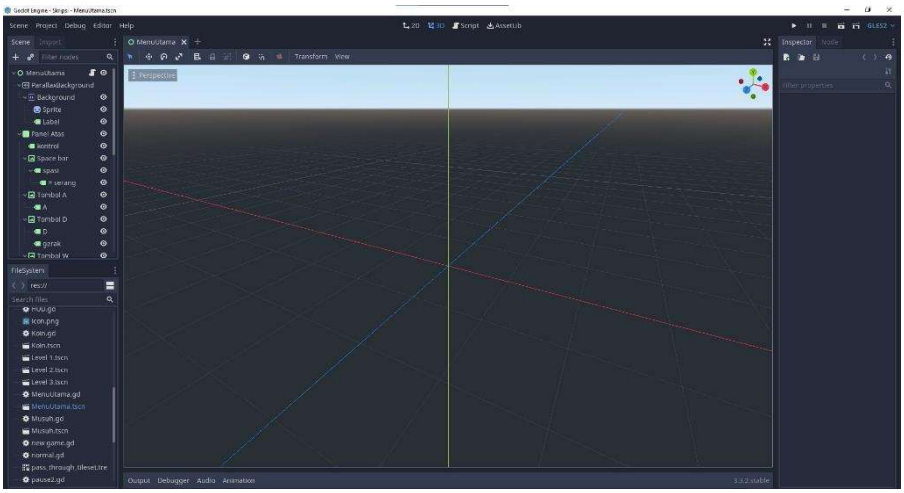
Video game yang memiliki jenis seperti *platformer* termasuk *game* yang mempersiapkan lingkungan yang baik dan bisa berbentuk dua dimensi ataupun tiga dimensi. *Video game* jenis ini memiliki tujuan untuk mengedalikan karakter utama sampai pada titik tujuan. Dalam *game genre* ini memiliki unsur yang paling penting, yaitu melompat. Permaianan ini memiliki mekanik dalam membimbing karakter pada pemain untuk melompati diantara rintangan dan plastform yang terdapat pada *genre* lain atau biasanya disebut sebagai *platforming* atau *platform* (Sulaeman & Permana, 2020).

Video game yang memiki jeni *platformer* ini memiliki aspek penting yaitu obejek yang berupa *platform*, seperti papan atau bidang yang mana objek lain atau karakter bisa berdiri atau dijadikan sebagai tempat untuk berjalan (Mustofa et al., 2018). Pemainan ini memiliki alur yaitu mengendalikan karakter utama untuk melompati atau melewati bidang yang pertama sampai dengan bidang yang terakhir. Selain itu juga terdapat firut lainnya yang telah disediakan, contohnya

objek yang dikumpulkan untuk menambah jumlah pada *score* (nilai), dan karakter yang lain seperti musuh akan menjadi hambatan dalam mengalahkan karakter utama.

2.2.4 Godot Engine

Yang dimaksud dengan Godot Engine yaitu suatu aplikasi dalam mengembangkan dan mempersiapkan berbagai *tools* pada *video game* yang nantinya akan dirilis secara *open source* dan gratis (*Godot Engine - Free and open source 2D and 3D game engine*, n.d.). Sistem *node* dan *scene* didukung dalam sistem ini. Setiap objek yang dikembangkan dalam *game* ini melambangkan sebagai *node*. *Node* bisa diletakkan dan digunakan kembali pada *scene* lainnya (Linden et al., 2020). *Engine* yang merancang *video game* ini akan sepenuhnya menjadi pemiliki dari pengembang itu. Pihak pemilik *engine* ini tidak akan meminta membayarkan keuntungan apabila pengembang *game* ini memiliki rencana untuk menjual *game* hasil buatannya sendiri. Apabila Godot Engine ini dibantu oleh pengguna *engine* dalam melakukan perkembangannya, dengan begitu pengguna bisa langsung mengujungi web dan memberikan donasi pada sejumlah yang sebagai sumbangan dalam membantu pengembangan dan perbaikan pada Godot Engine ini.



Gambar 2.5 Tampilan awal Godot Engine