BAB III

METODOLOGI DAN PERANCANGAN SISTEM

3.1. Metodologi Penelitian

Metodologi penelitian yang digunakan untuk penelitian ini antara lain adalah sebagai berikut:

a. Telaah literatur

Pada tahap literatur, dilakukan pengumpulan informasi. Informasi yang dikumpulkan berkaitan dengan chatbot Jacob, algoritma *Rule-based Lip-syncing*, *Hedonic Motivation System Adoption Model* (HMSAM) dan Skala Likert. Pengumpulan informasi dilakukan dengan membaca berbagai sumber seperti jurnal dan buku yang terkait dengan informasi yang dibutuhkan.

b. Analisis dan Perancangan

Pada tahap analisis dan perancangan, langkah pertama yang dilakukan adalah menganalisis *voice chatbot* Jacob yang telah ada, mulai dari fitur, cara pemasangan, cara penggunaan, dan bahasa pemrograman yang digunakan. Kemudian, dilakukan analisis terhadap cara menerapkan algoritma *Rule-based Lip-syncing* pada karakter visual Jacob. Setelah itu, dilakukan perancangan terhadap karakter virtual Jacob. Kemudian, dilakukan perancangan terhadap proses pengiriman message sebagai percakapan yang diucapkan oleh *chatbot* Jacob hingga fitur *lip-sync* pada karakter virtual Jacob dapat tercapai.

c. Implementasi

Pada tahap implementasi, dilakukan implementasi dari rancangan yang

dilakukan pada tahap analisis dan perancangan. Karakter virtual Jacob dikembangkan sebagai aplikasi *desktop*, yang terhubung dengan aplikasi *chatbot* Jacob yang berbasis web. Dalam pembuatan aplikasi desktop untuk karakter virtual Jacob, digunakan Unity versi 2019.2.15f1 dengan bahasa pemrograman C#. *Library* SpeechLib digunakan untuk melakukan *Speech Synthesis* (*text to speech*) dari *message* yang dikirim *chatbot* Jacob berbasis web agar dapat mengeluarkan output audio atau *speech* pada aplikasi desktop. Pengiriman message antara Jacob berbasis web dengan desktop dilakukan melalui bahasa pemrograman PHP dan database phpmyadmin.

d. Pengujian

Pada tahap pengujian, karakter virtual Jacob dan fitur *lip-sync* yang telah selesai diimplementasikan diuji. Dalam penelitian ini, digunakan uji coba dengan pendekatan *black box testing*, yang didasarkan pada detail aplikasi seperti tampilan, fungsi-fungsi yang ada, dan kesesuaian alur fungsi dengan bisnis proses yang diinginkan pengguna.

e. Evaluasi

Pada tahap evaluasi, dilakukan evaluasi terhadap fitur yang sudah ditambahkan pada *chatbot* Jacob. Evaluasi dilakukan melalui kuesioner yang telah diisi oleh pengguna dengan model *Hedonic Motivation System Adoption Model* (HMSAM), dengan menggunakan Skala Likert untuk mengetahui dan mengukur tingkat *behavioral intention to use* dan *immersion* terhadap karakter virtual Jacob yang memiliki fitur *lip-sync.* Pengguna akan menguji coba aplikasi secara langsung dan mengisi kuesioner yang disediakan dengan menggunakan model *Hedonic Motivation System Adoption Model* (HMSAM)

sehingga tingkat *behavioral intention to use* dan *immersion* dapat diketahui.

f. Penulisan Naskah Penelitian

Pada tahap ini, proses dan hasil akhir dari penelitian ini dituliskan dalam bentuk laporan sebagai bukti atas penelitian yang dilakukan.

3.2. Analisis Kebutuhan Fitur Lip-sync pada Virtual Karakter Jacob

Hasil analisis kebutuhan fitur *lip-sync* pada virtual karakter Jacob terdiri dari hasil analisis dari *voice chatbot* Jacob yang telah ada, dan kemudian, diikuti dengan hasil analisis terhadap cara menerapkan algoritma *Rule-based Lip-syncing* pada karakter visual Jacob. Berdasarkan analisis terhadap aplikasi *voice chatbot* Jacob yang telah ada, diketahui bahwa:

a. Bahasa yang digunakan sebagai *input* dan *output* merupakan bahasa Inggris.
b. Pertanyaan yang dapat diajukan pada *chatbot* Jacob adalah hal yang berkaitan dengan informasi Program Dual Degree Informatika UMN.
c. Proses berbicara antara *chatbot* Jacob dengan pengguna dimulai dari sapa, perkenalan nama (*chatbot* Jacob menanyakan nama dari pengguna), tanya jawab (pengguna dapat bertanya lebih dari satu kali), dan diakhiri dengan salam perpisahan.
d. Modul – modul *chatbot* Jacob yang ada saling terpisah, sehingga untuk menghubungkan antar modul digunakan API.
e. Bahasa pemrograman yang digunakan berbeda, yaitu PHP dan Python dikarenakan untuk pengembangan, lebih mudah menggunakan bahasa pemrograman PHP untuk aplikasi berbasis web, karena lebih familiar, dan kemudian *chatbot* Jacob menggunakan metode *machine learning* sehingga diperlukan penggunaan bahasa Python untuk dapat menggunakan *library*

machine learning.

Kemudian, analisis terhadap cara menerapkan algoritma *Rule-based Lip-syncing* pada karakter visual Jacob adalah sebagai berikut:

a. Karakter virtual *chatbot* Jacob adalah karakter tiga dimensi, bukan dua dimensi. Hal ini dikarenakan oleh beberapa hal. Pertama, bila menggunakan karakter tiga dimensi, pergerakan animasinya akan dapat dilihat dari segala posisi hanya dengan pengaturan posisi kamera, sehingga akan mempermudah pengembangan karakter Jacob apabila diperlukan pergerakan juga pada kepala dan badan Jacob. Kedua, karakter tiga dimensi memiliki *vertex* yang dapat disesuaikan dengan tulang dari karakter, dan *blend shapes* yang dapat diatur untuk pergerakan animasi pada karakter. *Blend shapes* dapat memiliki nilai dengan *range* tertentu, dan nilai dari *blend shapes* dapat diatur ataupun dikoding. Pada penelitian ini, *blend shapes* ini diperlukan untuk animasi pergerakan mulut karakter saat melakukan *lip-sync*.

b. Karakter virtual *chatbot* Jacob dibangun sebagai aplikasi desktop, sehingga terjadi pertukaran data antara *chatbot* Jacob berbasis web dengan aplikasi karakter virtual Jacob yang berbasis desktop.

c. Pembangunan aplikasi desktop untuk karakter virtual Jacob menggunakan Unity versi 2019.2.15f1 dengan bahasa pemrograman C#. Penggunaan aplikasi Unity dikarenakan terdapat versi gratisnya, mendukung pengembangan *multiplatform*, terdapat *community* yang suportif yang dapat memudahkan dalam pemecahan masalah, terdapat tutorial *online* yang memudahkan dalam mempelajari Unity. Oleh karena aplikasi karakter virtual Jacob menggunakan animasi tiga dimensi untuk melaksanakan fitur *lip-sync*,

maka Unity adalah aplikasi yang sesuai. Bahasa pemrograman yang digunakan adalah C# karena bahasa pemrograman yang dapat digunakan untuk membangun aplikasi menggunakan Unity adalah C#.

d. *Library* SpeechLib digunakan untuk melakukan *Speech Synthesis* (*text to speech*) dari *message* yang dikirim *chatbot* Jacob berbasis web agar dapat mengeluarkan output audio atau *speech* pada aplikasi desktop.

e. Pengiriman message antara Jacob berbasis web dengan desktop dilakukan melalui bahasa pemrograman PHP dan database phpmyadmin.

f. Dalam algoritma Rule-based Lip-syncing, sinyal audio diproses terlebih dahulu agar lip-sync dapat dilakukan. Pemrosesan sinyal audio dilakukan dengan sinyal audio di-window dengan Blackman window dan kemudian dihitung Fast Fourier Transform (FFT), proses ini dilakukan dengan menggunakan Scripting API dari Unity.

g. Rumus yang terdapat dari algoritma *Rule-based Lip-syncing* diterapkan dengan *coding* dalam bahasa pemrograman C#, sehingga dapat diintegrasikan dengan kodingan animasi karakter tiga dimensi dengan menggunakan aplikasi Unity.

3.3. Perancangan Sistem

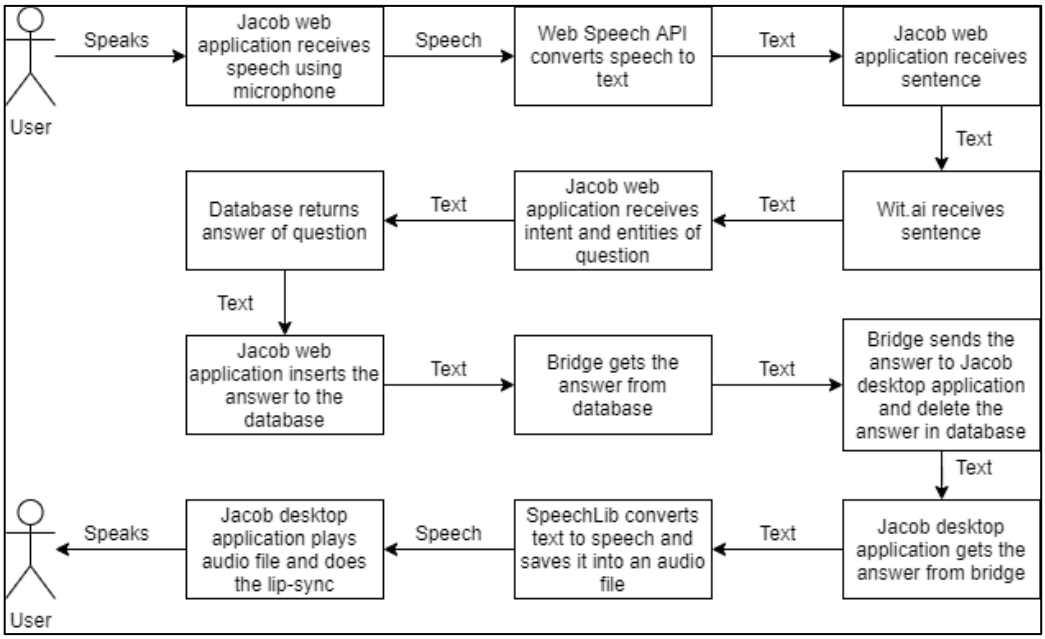
Dari hasil analisis yang telah dilakukan, maka dilakukan perancangan terhadap aplikasi karakter virtual Jacob, yang terkait dengan perancangan terhadap proses pengiriman *message* sebagai percakapan yang diucapkan oleh *chatbot* Jacob hingga fitur *lip-sync* pada karakter virtual Jacob dapat tercapai beserta antarmukanya.

Konsep pemrograman yang digunakan ada dua, yaitu *procedural* dan *Object Oriented Programming* (OOP). Proses penerimaan *message*, *speech synthesis*, visualisasi *lip-sync*, dan pengontrolan jalannya aplikasi menggunakan konsep pemrograman *procedural*. Sementara itu, proses *lip-sync* mulai dari analisis spektrum suara hingga *weight* untuk setiap *blend shapes* dan juga penyimpanan data informasi mengenai koneksi antara *blend shapes* yang dikoding dengan *blend shapes* pada karakter diprogram dengan menggunakan konsep pemrograman OOP.

Perancangan sistem dijelaskan dalam bentuk diagram alur kerja, diagram untuk menggambarkan pemrograman dengan konsep prosedural, seperti *flowchart*, diagram untuk menggambarkan pemrograman dengan konsep OOP, seperti *use case diagram*, *activity diagram*, *sequence diagram*, dan *class diagram*, serta diagram arsitektur, struktur tabel *database* tambahan untuk mengirim pesan dari Jacob berbasis web kepada Jacob berbasis desktop, dan rancangan antarmuka dari aplikasi karakter virtual Jacob.

3.3.1. Alur Kerja Aplikasi Karakter Virtual Jacob

Alur kerja aplikasi karakter virtual Jacob dimulai dari aplikasi Jacob berbasis web, di mana pengguna dapat bertanya pada *chatbot* Jacob melalui aplikasi Jacob berbasis web tersebut. Kemudian, jawaban diberikan melalui aplikasi karakter virtual Jacob yang berbasis desktop. Alur kerja aplikasi karakter virtual Jacob secara umum digambarkan pada Gambar 3.1.



Gambar 3.1. Alur kerja aplikasi karakter virtual Jacob secara umum

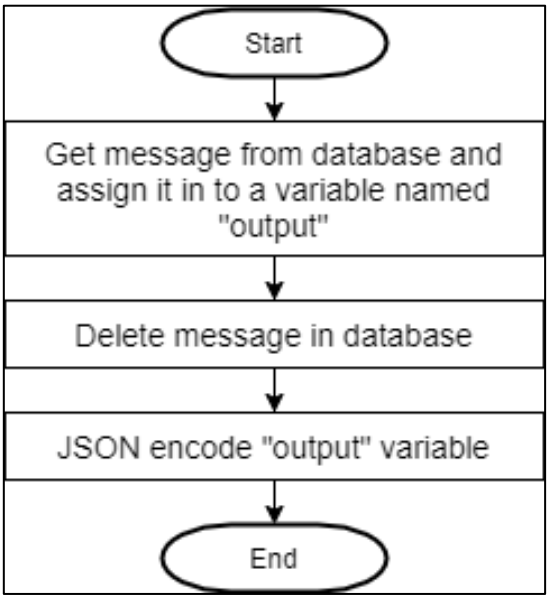
Pada Gambar 3.1, dapat dilihat bahwa pada langkah awal, pengguna memberikan pertanyaan kepada *chatbot* Jacob melalui aplikasi Jacob yang berbasis web. Setelah itu pemrosesan *speech* dari *user* dilakukan melalui aplikasi Jacob yang berbasis web hingga akhirnya Jacob mendapatkan jawaban yang akan menjadi *output*, yang berasal dari *database*. Ketika *chatbot* Jacob akan memberikan jawaban kepada *user*, jawaban tersebut dikirimkan ke aplikasi karakter virtual Jacob dengan cara jawaban disimpan di *database* terlebih dahulu oleh aplikasi berbasis web, kemudian diciptakan *bridge* yang menggunakan bahasa pemrograman PHP yang berfungsi sebagai jembatan antar aplikasi Jacob berbasis web dan desktop, agar dapat mengirim jawaban tersebut menuju aplikasi Jacob berbasis desktop dengan menggunakan metode JSON Encode.

Pada *bridge*, juga dilakukan penghapusan data *message* yang disimpan di *database*, karena data *message* tersebut hanya dilakukan untuk menampung jawaban agar dapat dikirimkan ke aplikasi desktop, sehingga akan dihapus terus

menerus agar *database* tidak menjadi penuh. Setelah itu, dengan menggunakan *library* SpeechLib, *text* dikonversi menjadi *speech*. *Speech* tersebut disimpan ke dalam *audio file*. *Audio file* tersebut diputar dan *lipsync* dilakukan oleh karakter virtual Jacob. Dengan begitu, *output* yang merupakan jawaban dari pertanyaan pengguna dikeluarkan melalui aplikasi karakter virtual Jacob.

3.3.2. Flowchart

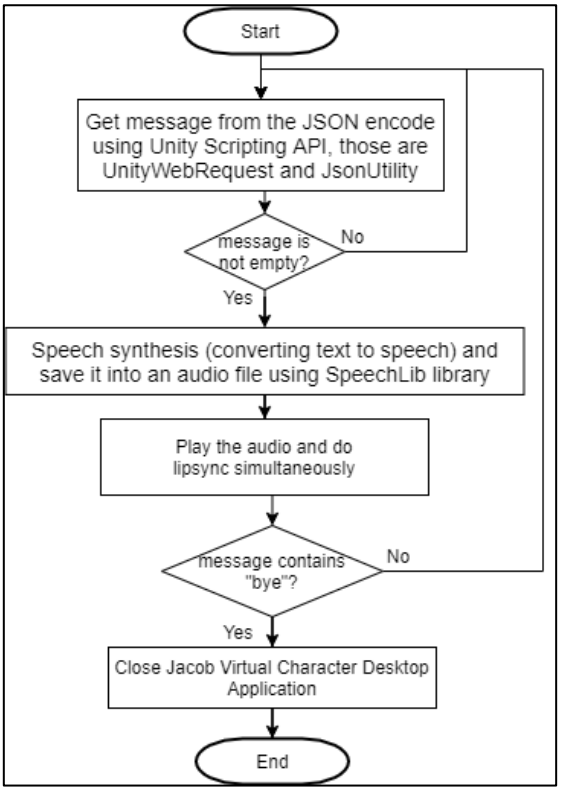
Sesuai yang dijelaskan pada alur kerja Jacob, aplikasi Jacob berbasis web membutuhkan bantuan *bridge* agar dapat mengirimkan *message* kepada aplikasi Jacob berbasis desktop. Aplikasi Jacob berbasis web akan menyimpan *message*, yang berupa jawaban atas pertanyaan pengguna, setiap kali Jacob akan menjawab pertanyaan pengguna. Lalu, dilakukan proses pada *bridge* di mana prosesnya dapat dilihat pada *flowchart* pada Gambar 3.2.



Gambar 3.2. *Flowchart bridge*

Pada Gambar 3.2, terlihat *bridge* mengambil data *message*, dan di-*assign* ke suatu variabel bernama *output*. Selanjutnya, *bridge* menghapus *message* yang terdapat pada *database*, agar data *message* kembali kosong, dan akan berisi kembali

apabila ada jawaban yang harus dikeluarkan pada pengguna. Hal ini dikarenakan data *message* di *database* hanya dijadikan tempat penampungan sementara. Lalu, *bridge* melakukan JSON Encode variabel *output*, yang berfungsi agar aplikasi karakter virtual Jacob yang berbasis desktop dapat menangkap variabel *output* pada *bridge*. Kemudian, aplikasi karakter virtual Jacob yang berbasis desktop mendapatkan *message* tersebut dan melakukan proses hingga terjadi *lip-sync*, yang prosesnya dapat dilihat pada *flowchart* pada Gambar 3.3.



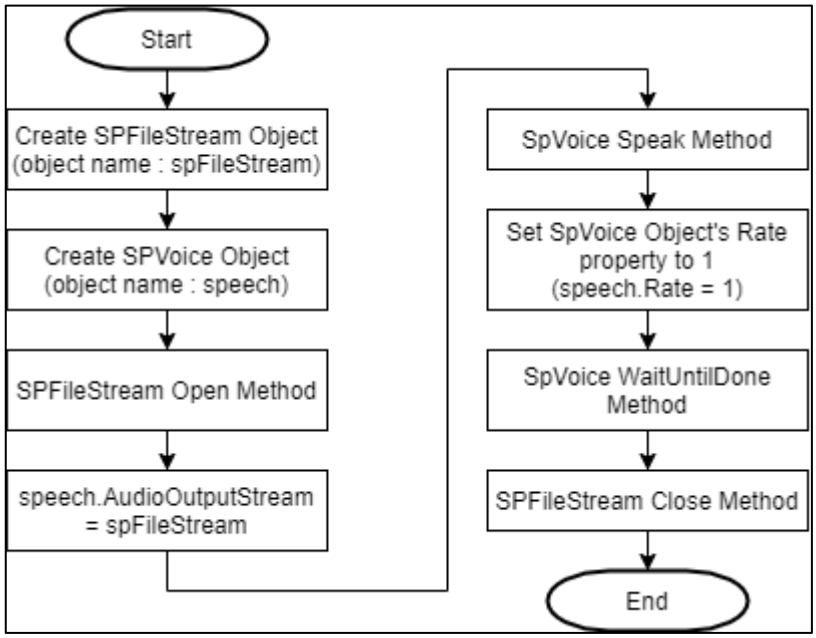
Gambar 3.3. *Flowchart* umum aplikasi karakter virtual Jacob berbasis desktop

Pada Gambar 3.3 terlihat bahwa aplikasi karakter virtual Jacob yang berbasis desktop mendapatkan *message* dengan menggunakan Scripting API dari Unity, yaitu UnityWebRequest dan JSONUtility. Apabila *message* yang diterima masih kosong, maka *request* akan dilakukan terus menerus hingga didapatkan *message* yang tidak kosong. *Looping* ini bertujuan agar aplikasi karakter virtual Jacob dapat

mengeluarkan apabila ada jawaban yang harus dikeluarkan pada pengguna, namun aplikasi karakter virtual Jacob akan diam bila belum saatnya untuk memberikan jawaban. Data *message* akan kosong apabila tidak ada yang harus di-*output*-kan lagi, dan berisi apabila ada jawaban yang harus disampaikan pada pengguna.

Setelah itu, apabila terdapat jawaban yang harus dikeluarkan, dilakukan *speech synthesis* atau pengkonversian dari *text* menjadi *speech* dengan menggunakan library SpeechLib, dan disimpan dalam bentuk *audio file*. Setelah itu, *audio file* diputar, dan *lip-sync* pada karakter dilakukan secara bersamaan. Audio harus disimpan ke dalam suatu *audio file* dikarenakan untuk melakukan fitur *lip-sync*, harus dilakukan pemutaran *audio* agar spektrum frekuensi yang didapatkan lebih bagus. Apabila *message* mengandung kata "*bye*", maka aplikasi akan tertutup.

Proses *speech synthesis* hingga hasil *speech synthesis* disimpan ke dalam *audio file*. Prosesnya dapat dilihat pada *flowchart* yang terdapat pada Gambar 3.4.



Gambar 3.4. *Flowchart Speech Synthesis*

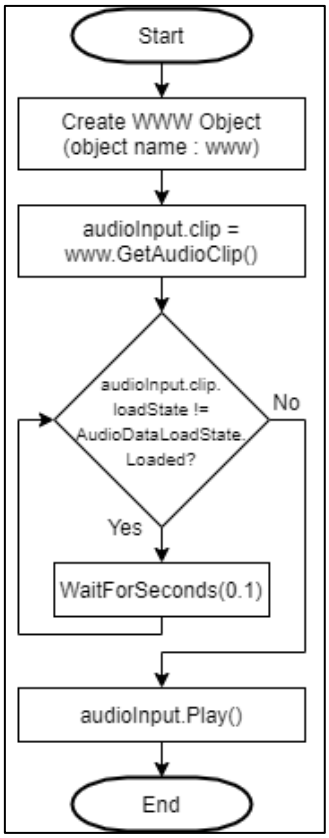
Pada Gambar 3.4, diperlihatkan bahwa langkah awal dimulai dari membuat *object* SPFileStream, kemudian membuat *object* SPVoice. Object SPFileStream dan SPVoice berasal dari library SpeechLib, dan SpeechLib berasal dari Microsoft Speech API. SPFileStream berguna agar *data streams* dapat di-*read* dan *write* sebagai *file*. SPVoice berguna untuk membawa kapabilitas *text to speech engine* ke aplikasi dengan menggunakan otomasi Microsoft Speech API.

Langkah selanjutnya adalah SPFileStream Open Method, yang berfungsi untuk membuka *filestream object* sesuai dengan *path* yang ditentukan untuk *reading* dan *writing.* Selanjutnya, properti AudioOutputStream dari objek SPVoice di-*assign* dengan objek dari SPFileStream. AudioOutputStream di-*set* agar format *output audio* diubah secara otomatis sesuai dengan format *output audio* yang sesuai dengan mesin *text-to-speech* (TTS), di mana untuk aplikasi karakter virtual Jacob, format *output audio* dari *text-to-speech* yang dipilih adalah berupa *wav* file.

Kemudian, dilanjutkan dengan SPVoice Speak Method, yaitu *method* untuk *speaking.* Pada aplikasi karakter virtual Jacob, Speak Method dipanggil secara *asynchronous* agar proses *speak* dilakukan sebagai *background process.* Kemudian, properti *rate* dari objek SPVoice diset, untuk mengatur kecepatan bicara.

Lalu, dilanjutkan dengan SPVoice WaitUntilDone Method, yang berfungsi untuk memblokir pemanggilan Speak Method hingga proses *speak* selesai, namun sementara itu *user* masih bisa melakukan interaksi dengan *mouse* ataupun *keyboard*. Setelah *speak* berhasil di-*write* ke *audio file*, maka dilakukan SPFileStream Close Method yang berguna untuk menutup objek *filestream*.

Proses pemutaran audio dijelaskan pada *flowchart* yang terdapat pada Gambar 3.5.



Gambar 3.5. *Flowchart* pemutaran *audio*

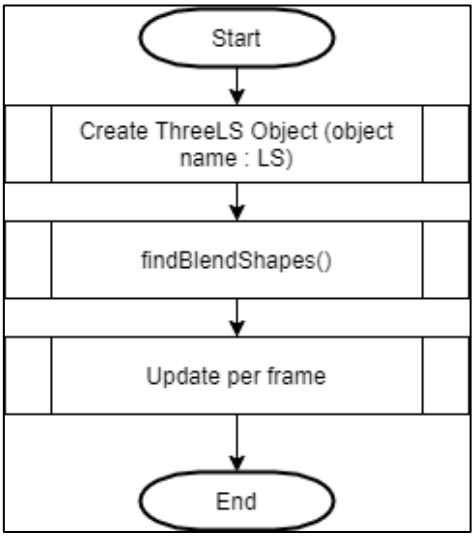
Pada Gambar 3.5, terlihat bahwa langkah pertama untuk pemutaran *audio* dimulai dari pembuatan *object* WWW, di mana WWW adalah sebuah *class* dari UnityEngine, yang bertujuan untuk mengambil konten dari URL. Kemudian, di langkah berikutnya terdapat variabel audioInput, di mana variabel audioInput merupakan objek dari *class* AudioSource.

AudioSource merupakan *class* dari UnityEngine yang berfungsi sebagai representasi dari *audio source* dalam 3D. Properti *clip* pada AudioSource berfungsi untuk menentukan *audio clip* yang akan diputar. *Method* GetAudioClip pada *class* WWW berfungsi untuk mengambil konten *audio clip* dari suatu URL yang ditentukan.

Kemudian, dilanjutkan dengan *looping*, di mana apabila audio belum dapat dimuat, maka akan dilakukan penungguan selama 0,1 detik, dan dilanjutkan

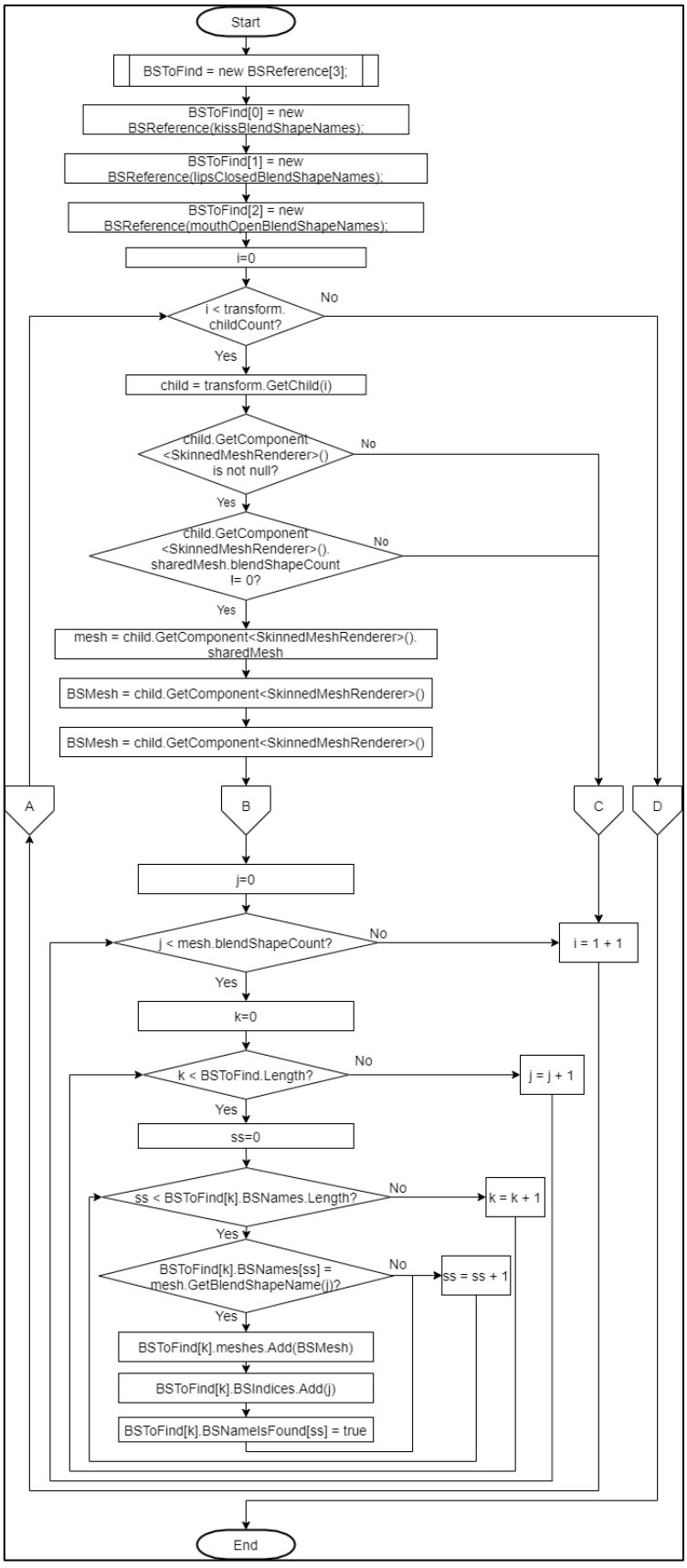
looping, hingga audio sudah dapat dimuat. Properti loadState dari *audio clip* mengembalikan *current load state* dari *audio clip,* sedangkan AudioDataLoadState.Loaded merupakan nilai yang dikembalikan oleh *property* loadstate dari *audio clip* yang telah berhasil memuat data audionya. Apabila *audio* telah berhasil dimuat, maka nilai dari audioInput.clip.loadState dan AudioDataLoadState.Loaded akan sama. Setelah itu, dijalankan *method* Play dari *class* AudioSource, yang berguna untuk memutar *audio.*

Proses *lip-sync* dilakukan bersamaan dengan pemutaran *audio*. Proses terjadinya *lip-sync* dapat dilihat pada Gambar 3.6.



Gambar 3.6. *Flowchart lip-sync* secara umum

Pada Gambar 3.6, dapat diketahui bahwa langkah awal dimulai dari pembuatan objek ThreeLS. Kemudian, dilanjutkan dengan menjalankan modul findBlendShapes yang prosesnya dapat dilihat pada *flowchart* pada Gambar 3.7. Lalu, dilanjutkan dengan modul *update* yang melakukan *update per frame,* yang prosesnya dapat terlihat pada *flowchart* pada Gambar 3.8.



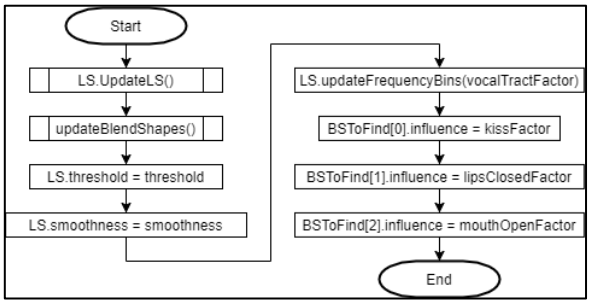
Gambar 3.7. *Flowchart* modul findBlendShapes

Modul findBlendShapes berfungsi untuk mencari *blend shapes* pada karakter virtual Jacob untuk melakukan *lip-sync*. Proses yang dilakukan dimulai dengan langkah awal, yaitu membuat objek dari *class* BSReferences, yang bertujuan untuk menyimpan data mengenai koneksi antara *blend shapes* yang dikoding dengan *blend shapes* pada karakter. Lalu, dilanjutkan dengan *looping* untuk mencari *children* yang memiliki *blend shapes.*

Di dalam *looping* untuk mencari *children* yang memiliki *blend shapes*, terdapat *looping* untuk mencari korespondensi dari *blend shapes*. Kemudian, di dalamnya terdapat *looping* untuk menyimpan referensi dari *mesh* dan *index* dari *blend shapes*, melalui iterasi dari bentuk *kiss, lips closed,* dan *mouth open*.

Di dalam *looping* untuk menyimpan referensi dari *mesh* dan *index* dari *blend shapes*, terdapat *looping* sebagai iterasi dari nama *blend shapes* yang di-*assign* ke setiap *blend shape* untuk *lip-sync*. Di dalamnya terdapat pilihan jika nama *blend shape* dari suatu *mesh* sama dengan nama *blend shape* yang dideklarasikan pada Unity Editor, maka simpan *mesh* (meshes), BS *index* (BSIndices), dan variabel yang menunjukkan *blend shape* telah ditemukan (BSNameIsFound).

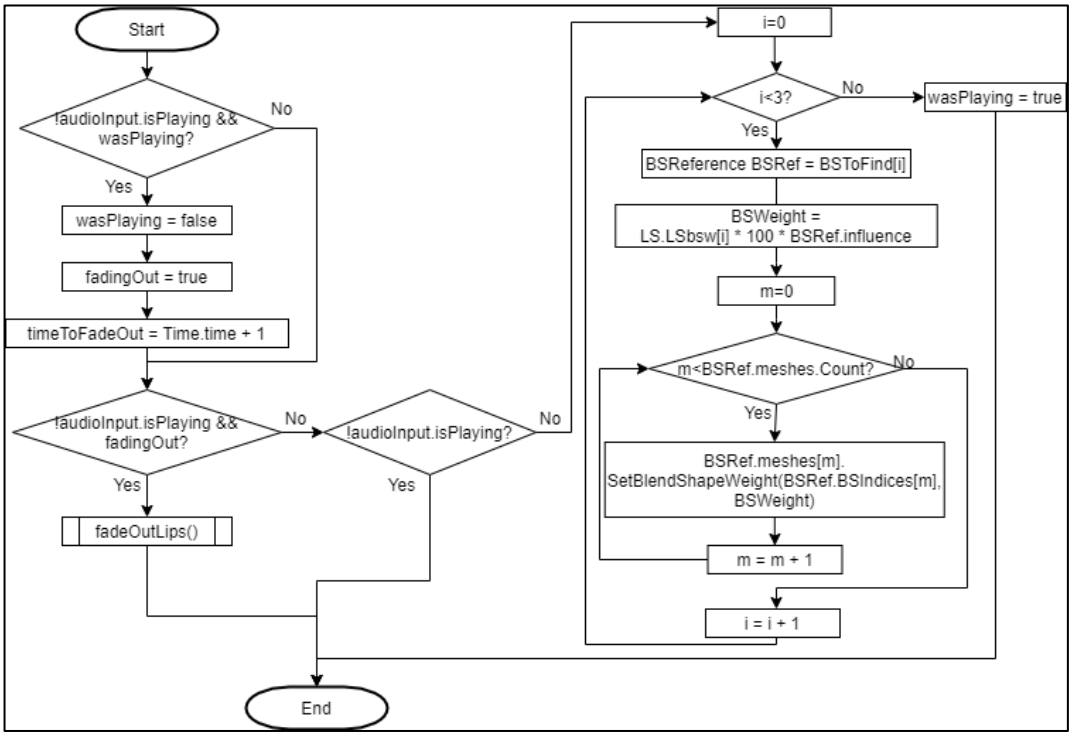
Sesudah modul findBlendShapes dijalankan, maka dilakukan *update per frame*. Proses pada modul *update per frame* dijelaskan pada *flowchart* yang terlihat pada Gambar 3.8.



Gambar 3.8. *Flowchart* modul *update per frame*

Modul *update per frame* digunakan untuk melakukan *update per frame*, agar dapat terjadi animasi. Pada Gambar 3.8, dapat dilihat bahwa langkah *update* yang dilakukan per *frame* untuk yang pertama adalah menjalankan *method* UpdateLS dari *class* ThreeLS.

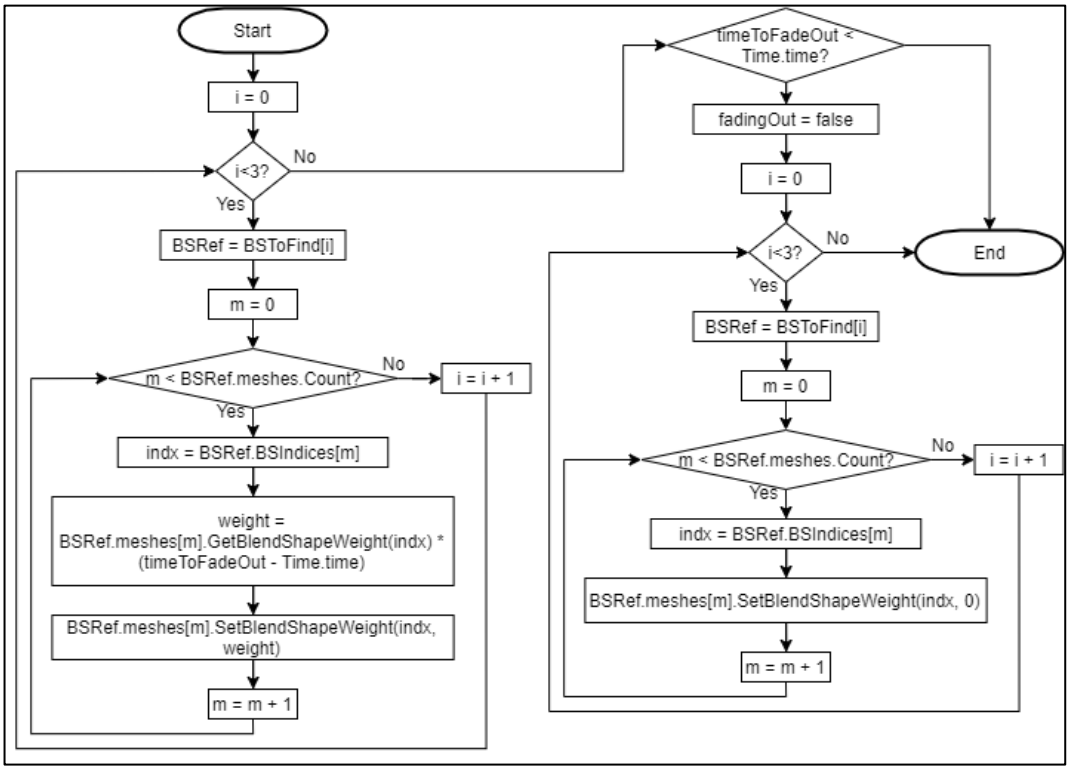
Lalu, dilanjutkan dengan pelaksanaan modul updateBlendShapes, yang prosesnya dapat dilihat pada *flowchart* pada Gambar 3.9. Kemudian dilanjutkan dengan set *property threshold* dan *smoothness* pada variabel LS. Lalu, dijalankan *method* updateFrequencyBins. Selanjutnya, dilakukan *assign* terhadap variabel influence pada BSToFind.



Gambar 3.9. *Flowchart* modul updateBlendShapes

Pada Gambar 3.9, terlihat proses yang dilakukan pada modul updateBlendShapes, di mana modul ini berfungsi untuk *update* nilai dari *blend shapes kiss*, *lips closed*, dan *mouth open*. Dimulai dari pengecekkan apabila audio berhenti, maka akan dilakukan *fading out* dalam waktu 1 *second*.

Apabila nilai *fading out* bernilai *true*, maka akan dijalankan modul fadeOutLips yang berfungsi untuk mengembalikan nilai dari *blend shape* pada bibir menjadi 0 sehingga mulut akan kembali ke bentuk semula ketika tidak berbicara. Proses fadeOutLips dijelaskan pada *flowchart* pada Gambar 3.10. Setelah itu, dilakukan *looping* untuk mengisi nilai *weight* dari masing – masing *blend shape* kiss, lipClosed, dan mouthOpen.

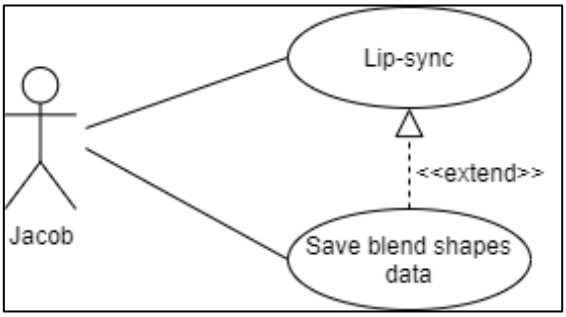


Gambar 3.10. Flowchart modul fadeOutLips

Proses fadeOutLips berfungsi untuk *fade out* nilai *blend shapes* menjadi 0. Dimulai dengan *looping* awal yang berguna untuk mengurangi nilai dari *blend shapes* secara perlahan secara linear. Kemudian, apabila timeToFadeOut lebih kecil daripada Time.time, yang merupakan waktu lamanya aplikasi berjalan, maka dilakukan nilai dari fadingOut di-set kembali menjadi *false*, dan dilakukan *looping* untuk *set* nilai dari setiap *blend shapes* menjadi 0.

3.3.3. Use Case Diagram

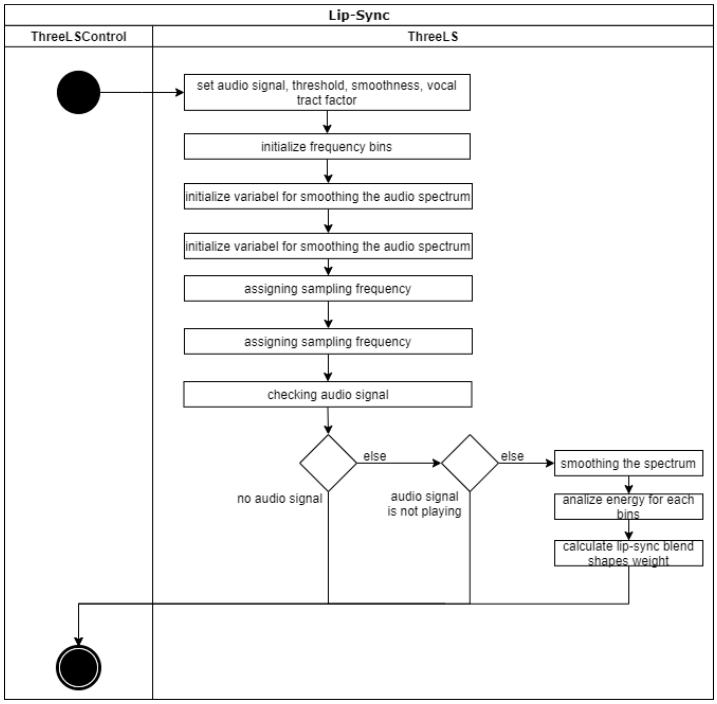
Use case diagram yang terlihat pada Gambar 3.11 menggambarkan fungsionalitas yang terdapat pada aplikasi karakter virtual Jacob, yang diprogram dengan menggunakan konsep pemrograman OOP. Pada *use case diagram*, terdapat *actor* berupa Jacob, dan *use case* terdiri dari dua, yaitu *lip-sync* dan *save blend shapes data*. Jacob melakukan proses untuk menghasilkan *lip-sync* dengan menganalisis spektrum suara hingga mendapatkan *weight* untuk setiap *blend shapes*. Jacob juga menyimpan data mengenai *blend shapes*, yaitu data mengenai koneksi antara *blend shapes* yang dikoding dengan *blend shapes* pada karakter.



Gambar 3.11. *Use Case Diagram* Aplikasi Karakter Virtual Jacob

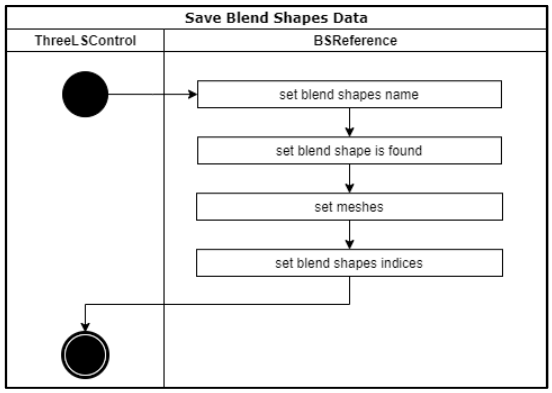
3.3.4. Activity Diagram

Activity diagram pada aplikasi karakter virtual Jacob dibagi berdasarkan *use case* yang ada, sehinga terdapat *activity diagram* mengenai *lip-sync* dan mengenai penyimpanan data tentang *blend shapes*. A*ctivity diagram* mengenai *lip-sync* dapat dilihat pada Gambar 3.12. Proses *lip-sync* dilaksanakan pada *class* ThreeLS, yang objeknya dibuat pada ThreeLSControl, yang mengontrol jalannya aplikasi karakter virtual karakter Jacob termasuk penerimaan *message*, *speech synthesis*, dan visualisasi *lip-sync*. Proses *lip-sync* dilaksanakan sesuai dengan algoritma Rule-based Lip-syncing, yang rumusnya terdapat pada Bab 2.



Gambar 3.12. *Activity diagram lip-sync*

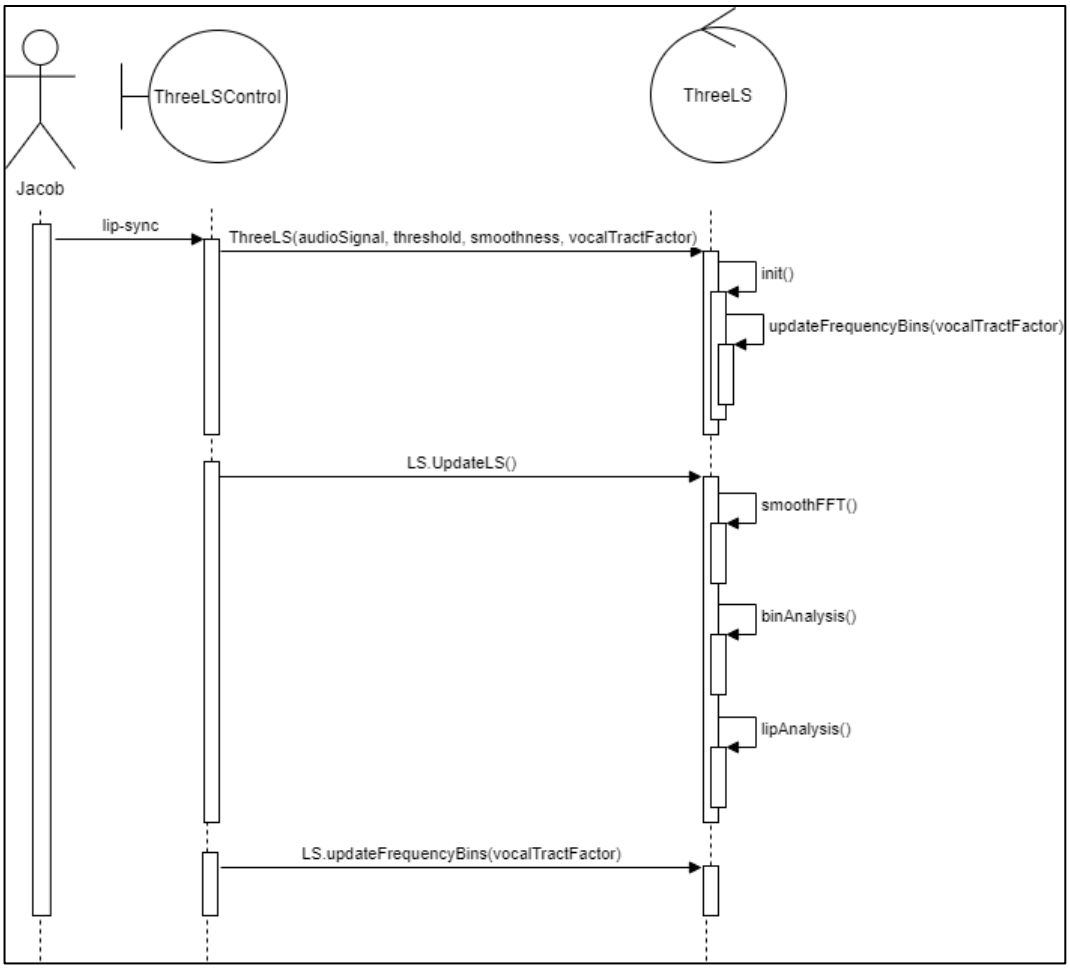
Activity diagram mengenai data tentang *blend shapes* dapat dilihat pada Gambar 3.13. Data tentang *blend shapes* berupa informasi mengenai koneksi antara *blend shapes* yang dikoding dengan *blend shapes* yang terdapat pada karakter. Variabel yang di-*set* adalah nama *blend shapes*, variabel yang menunjukkan apakah *blend shapes* ditemukan, variabel untuk *meshes*, dan variabel untuk *index blend shapes*. Prosesnya dilaksanakan pada *class* BSReference, dan objeknya dibuat pada ThreeLSControl.



Gambar 3.13. *Activity diagram save blend shapes data*

3.3.5. Sequence Diagram

Sequence Diagram pada karakter virtual Jacob dibagi berdasarkan *use case diagram* yang ada, sehingga terdapat *sequence diagram* mengenai *lip-sync* dan mengenai penyimpanan data tentang *blend shapes. Sequence diagram* mengenai *lip-sync* dapat dilihat pada Gambar 3.14.



Gambar 3.14. *Sequence diagram lip-sync*

Seperti yang terlihat pada Gambar 3.14, proses *lip-sync* dimulai dengan pembuatan objek ThreeLS dengan menggunakan *constructor*-nya. Pada *constructor* ThreeLS dilakukan langkah *assign* terhadap variabel audioSignal, threshold, smoothness, dan vocalTractFactor. Kemudian, dilanjutkan dengan

menjalankan metode init, yang merupakan *method* dari *class* ThreeLS yang berfungsi untuk inisialisasi.

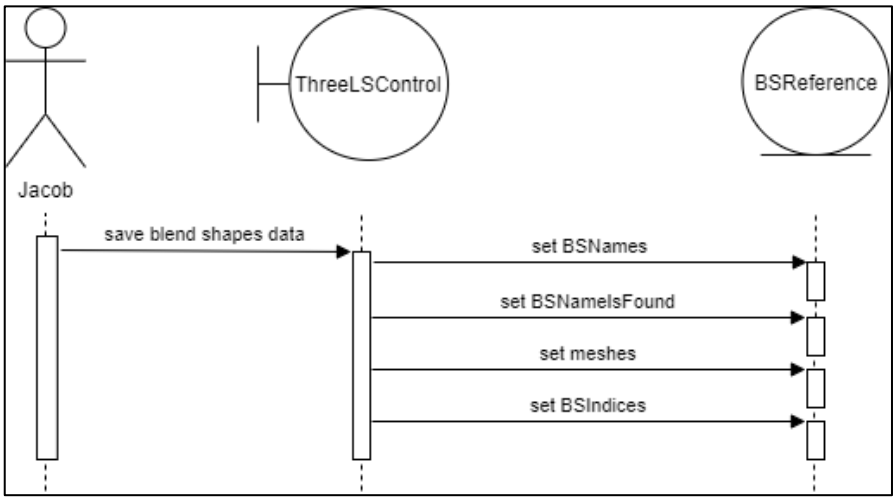
Langkah pertama yang dilakukan oleh *method* init adalah menjalankan *method* dari *class* ThreeLS, yaitu updateFrequencyBins untuk menginisialisasi *frequency*. Kemudian, dilakukan inisialisasi FFT, dan *assign* terhadap variabel fs, yang merupakan variabel untuk *sampling frequency*.

Saat terjadi *update per frame*, dilakukan pemanggilan method UpdateLS. Pada *method* UpdateLS, langkah awal yang dilakukan adalah pengecekan, yang apabila terdapat *signal* audio dan audio sedang diputar, maka dijalankan *method* smoothFFT, binAnalysis, dan lipAnalysis, yang merupakan *method* pada *class* ThreeLS.

Pada *method* smoothFFT, langkah pertama yang dilakukan adalah mengambil data *spectrum* dari audio yang ada, dengan di-*window* dengan Blackman Window dan kemudian menghitung Fast Fourier Transform. Kemudian, dilakukan *looping* untuk *smoothing spectrum* yang ada menggunakan Rumus 2.1 dan dikonversikan menjadi satuan dB menggunakan Rumus 2.2. *Method* binAnalysis berguna untuk melakukan perhitungan pada *energy* tiap *bins*, sesuai dengan Rumus 2.3, Rumus 2.5, dan Rumus 2.6. *Method* lipAnalysis berfungsi untuk menghitung *weight* dari setiap *lip-sync blend shapes.*

Saat terjadi *update per frame*, dilakukan juga pemanggilan *method* updateFrequencyBins. Method updateFrequencyBins berfungsi untuk melakukan *update* pada nilai *frequency bins* dengan melakukan skalasi dengan *vocal tract factor*, sesuai dengan Rumus 2.4.

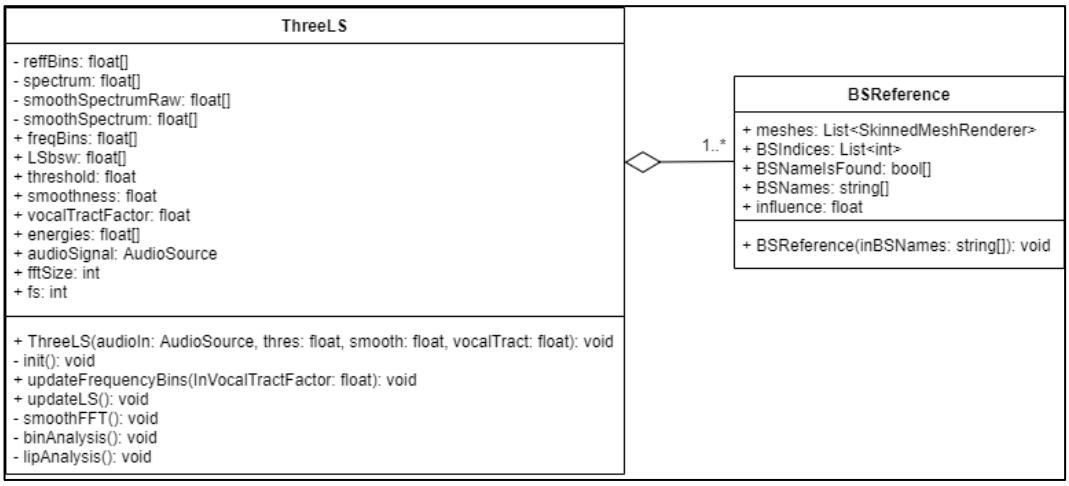
Sequence diagram mengenai penyimpanan data tentang *blend shapes* dapat dilihat pada Gambar 3.15. Ketika ingin menyimpan data tentang *blend* shapes, dilakukan pemanggilan *constructor* BSReference, yang pada prosesnya dilakukan *assign* terhadap variabel BSNames, BSNameIsFound, meshes, dan BSIndices.



Gambar 3.15. *Sequence diagram save blend shapes data*

3.3.6. Class Diagram

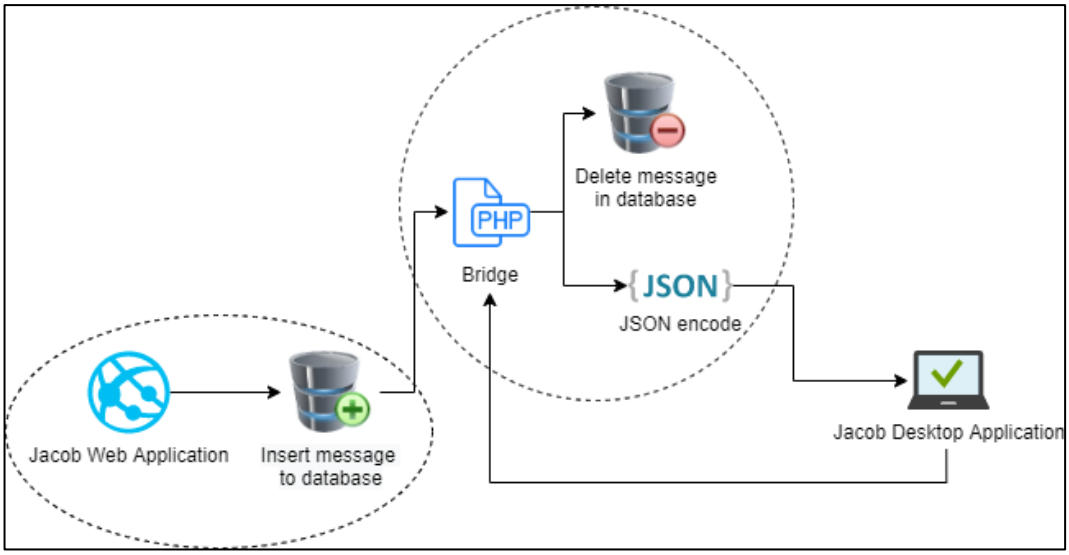
Dalam memprogram aplikasi karakter virtual Jacob dengan konsep pemrograman OOP, terdapat *class* ThreeLS dan *class* BFReference. *Property* dan *method* serta hubungan kedua *class* ini dapat dilihat pada Gambar 3.16.



Gambar 3.16. *Class Diagram* Aplikasi Karakter Virtual Jacob

3.3.7. Diagram Arsitektur

Diagram arsitektur dari aplikasi karakter virtual Jacob digambarkan pada Gambar 3.17. Pada diagram arsitektur Jacob terlihat bahwa dibutuhkan tiga komponen utama, yaitu Jacob Web Application, Bridge, dan Jacob Desktop Application. Jacob Web Application memasukkan data *message* ke *database*. Kemudian, pada *bridge*, digunakan *file* berekstensi php untuk mengirim data *message* ke Jacob Desktop Application dengan mengeluarkan *output* JSON Encode, dan *bridge* juga berperan dalam penghapusan data *message* pada *database*. Desktop Application mengirim *request* ke *bridge*, dan mendapatkan data *message* dari JSON encode tersebut.



Gambar 3.17. Diagram Arsitektur Aplikasi Karakter Virtual Jacob

3.3.8. Struktur Tabel Database Tambahan untuk Aplikasi Karakter Virtual Jacob

Pada penelitian sebelumnya, telah digunakan database untuk menyimpan data mengenai pertanyaan, intent, entities, logs, admin, dan lainnya. Namun, di dalam penelitian ini dilakukan penyesuaian dengan *database* Jacob sebelumnya.

Penyesuaian tersebut dilakukan dengan penambahan tabel, yaitu tabel sendUnity. Struktur tabel sendUnity dapat dilihat pada Tabel 3.1.

Tabel 3.1. Struktur tabel sendUnity

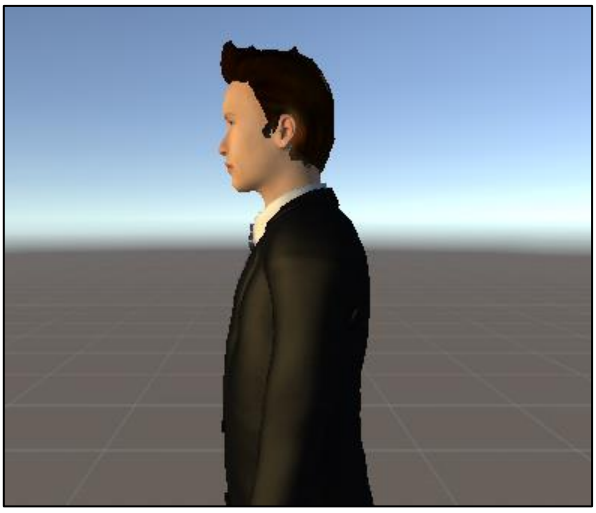
Field	Type	Null	Key	Default	Description
message	Varchar(255)	No	-	None	Berisi message yang akan diucapkan oleh Jacob

Berdasarkan Tabel 3.1, dapat dilihat bahwa tabel sendUnity berisikan kolom *message*, yang berfungsi untuk menyimpan *message* yang akan diucapkan oleh Jacob.

3.3.9. Rancangan Antarmuka Aplikasi Karakter Virtual Jacob

Karakter virtual Jacob dirancang menggunakan MakeHuman, dengan menggunakan aset yang disediakan oleh MakeHuman dan *community* MakeHuman dengan beberapa penyuntingan warna. Untuk *blend shape* dari karakter Jacob dibuat menggunakan aplikasi Blender. Karakter Jacob digambarkan pada Gambar 3.18, Gambar 3.19, dan Gambar 3.20.

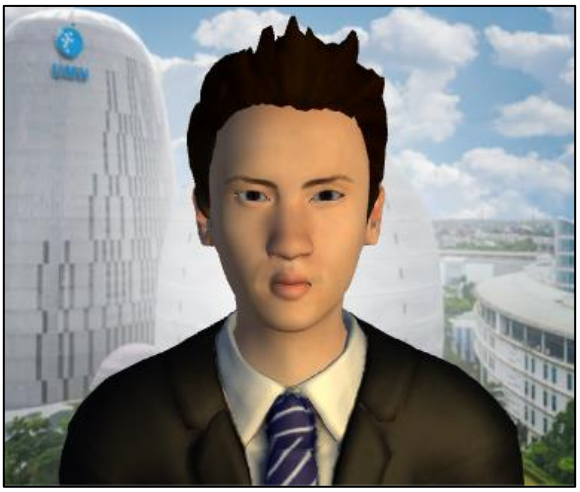
Gambar 3.18. Karakter Jacob tampak depan setengah badan



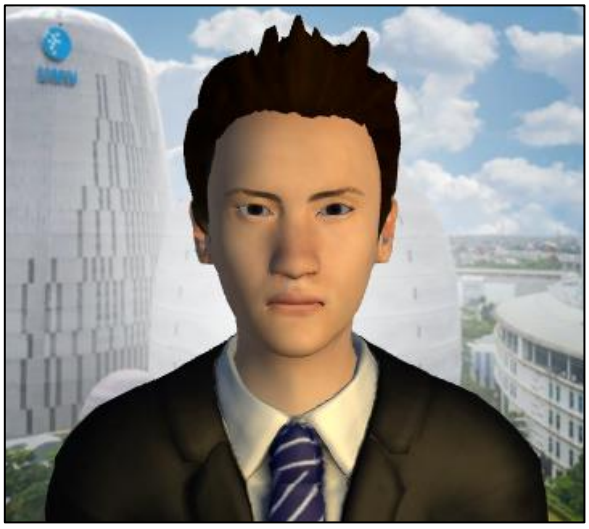
Gambar 3.19. Karakter Jacob tampak samping

Gambar 3.20. Karakter Jacob tampak depan satu badan

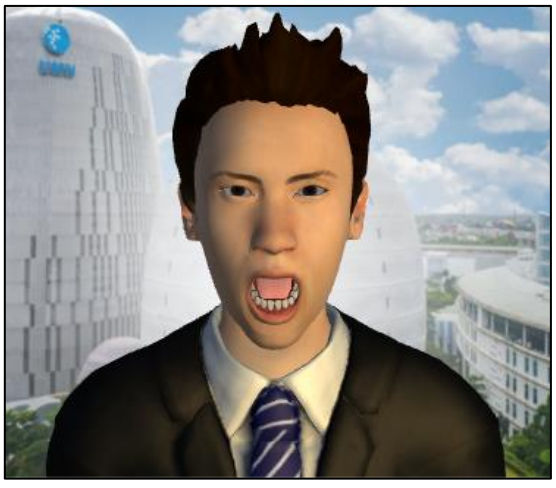
Wajah karakter virtual Jacob ketika dikenakan blend shape kiss, lips closed, dan mouth open adalah seperti pada Gambar 3.21, Gambar 3.22, dan Gambar 3.23.



Gambar 3.21. Jacob *kiss blend shape*



Gambar 3.22. Jacob *lips closed blend shape*



Gambar 3.23. Jacob *mouth open blend shape*

Aset yang diperlukan untuk membangun karakter virtual Jacob adalah sebagai berikut:

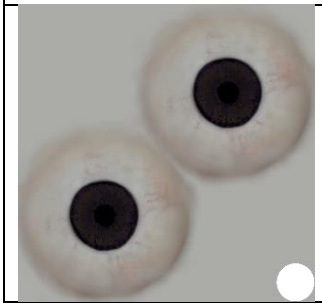
a. Material

Tabel 3.2. Material karakter virtual Jacob

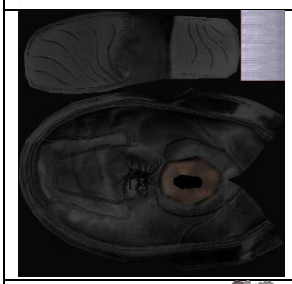
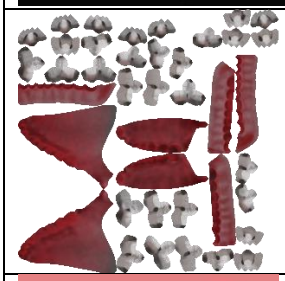
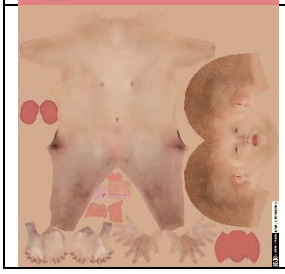
Gambar	Nama Material	Keterangan	Sumber
	Eyebrow 008	Alis mata	MakeHuman
	High-Poly	Bola Mata	MakeHuman
	Male ElegantSuit 01	Pakaian	MakeHuman
	Shoes 04	Sepatu	MakeHuman
	Male Generic	Badan	MakeHuman
	Elvs Maxwell Hair	Rambut	Elvaerwyn (MakeHuman Community)
	Teeth Base	Gigi	MakeHuman
	Tongue 01	Lidah	MakeHuman

b. Texture

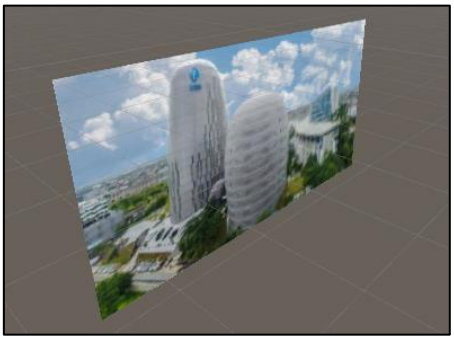
Tabel 3.3. Texture karakter virtual Jacob

Gambar	Keterangan	Sumber
	Texture untuk mata	MakeHuman, dengan penyuntingan warna

Tabel 3.3. Texture karakter virtual Jacob (lanjutan)

Gambar	Keterangan	Sumber
	Texture untuk pakaian	MakeHuman, dengan penyuntingan warna
	Texture untuk rambut	Elvaerwyn (MakeHuman Community), dengan penyuntingan warna
	Texture untuk sepatu	MakeHuman
	Texture untuk gigi	MakeHuman
	Texture untuk lidah	MakeHuman
	Texture untuk badan	MakeHuman

Latar belakang dari karakter virtual Jacob dibuat dengan menggunakan quad. Quad merupakan material yang merepresentasikan *flat surfaces*. *Surface* pada quad menggunakan foto Universitas Multimedia Nusantara sehingga karakter virtual Jacob memiliki latar belakang berupa foto Universitas Multimedia Nusantara. Tampilan quad untuk latar belakang karakter virtual Jacob terlihat seperti pada Gambar 3.24.



Gambar 3.24. Latar belakang karakter virtual Jacob

Aset yang diperlukan untuk membangun latar belakang untuk tampilan karakter virtual Jacob adalah sebagai berikut:

a. Material

   Material yang digunakan adalah Quad, yang merupakan objek 3D yang tersedia dari Unity.

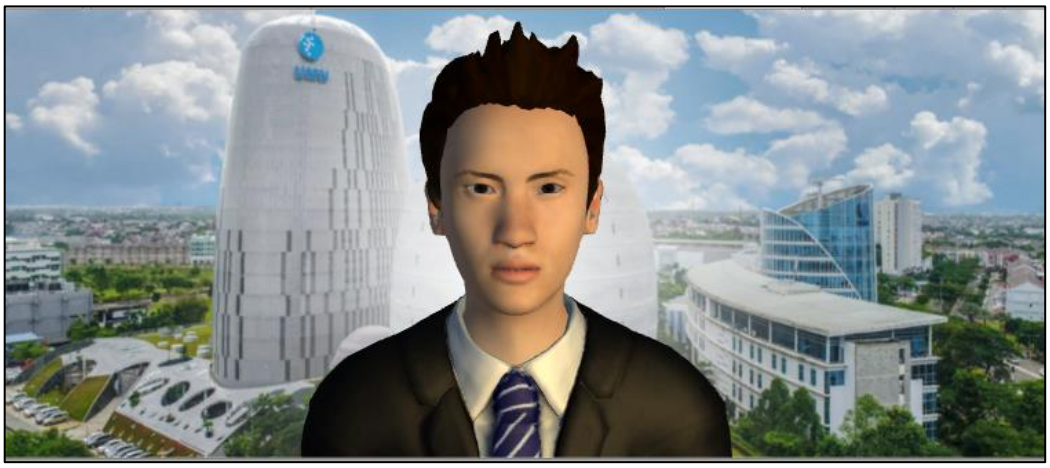
b. Texture

   Texture yang digunakan adalah gambar, seperti terlihat pada Gambar 3.25.

Gambar 3.25. *Texture* latar belakang karakter virtual Jacob

(Universitas Multimedia Nusantara, 2019)

Rancangan antarmuka untuk aplikasi karakter virtual Jacob terdiri dari karakter Jacob, yang di belakangnya terdapat latar belakang berupa foto Universitas Multimedia Nusantara. Rancangan antar muka aplikasi karakter virtual Jacob digambarkan seperti pada Gambar 3.26.



Gambar 3.26. Rancangan antarmuka aplikasi karakter virtual Jacob