BAB II

LANDASAN TEORI

2.1. Tinjauan Jurnal

Sejarah tentang *Augmented Reality* dimulai dari tahun 1957-1962, ketika seorang penemu yang bernama Morton Heilig, seorang *sinematografer*, menciptakan dan mematenkan sebuah simulator yang disebut Sensorama dengan visual, getaran dan bau.

Menurut Jacko, dkk (2010). Augmented Reality (AR), adalah "teknologi yang menggabungkan benda maya dua dimensi dan ataupun tiga dimensi kedalam sebuah lingkungan nyata tiga dimensi lalu memproyeksikan benda-benda maya tersebut dalam waktu nyata"

Sedangkan Menurut Butchart (2011:1-2) "*Augmented Reality* dapat memperkaya perspektif dengan memasang objek virtual kedalam dunia nyata dengan cara sebagaimananya sehingga yang melihatnya seakan-akan objek virtual tersebut adalah bagian dari lingkungan nyata"

Maksudnya *augmented reality* (AR) adalah aplikasi yang dapat menggabungkan objek maya dengan lingkungan nyata secara waktu nyata atau *real time* dalam satu frame camera

Menurut Candra, dkk (2012). "Aplikasi yang dibangun mampu mengenali marker dan dapat menampilkan video yang di-load melalui URL"

Sedangkan Menurut Chung, (2008). "Hal tersebut (*augmented reality*) tentunya akan membuat user lebih aktif dalam mengolah informasi dan respon *user* terhadap suatu barang juga akan menjadi lebih baik"

Teknologi *augmented reality* dapat dimanfaatkan untuk merancang sebuah konsep perpanjangan informasi dari media promosi cetak ke media promosi berbentuk video, atau gambar sehingga bisa menarik perhatian konsumen menggunakan terhadap suatu produk karna bersifat *interaktif*

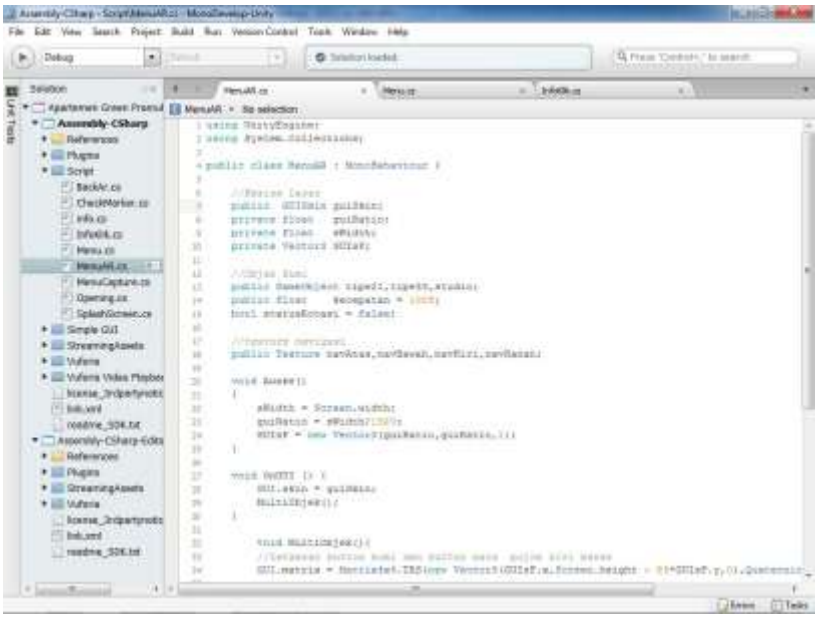
Menurut Adhi, dkk (2010) "Dalam kurun waktu 2005 hingga 2009 minat orang akan AR (*augmented reality*) sangat tinggi. Ini terlihat dari frekuensi *google search* diinternet yang banyak mengakses informasi mengenai *augmented reality* (AR) "

Menurut Edi, dkk (2010). " *augmented reality* (AR) telah banyak digunakan didunia hiburan, pelatihan militer, medis, desain rekayasa, robotik dan telerobotik, manufaktur, pendidikan, dan lain-lain"

2.2. Konsep Dasar Program

a. Bahasa C#

Proses ini memberikan fungsi pada aplikasi dengan cara memberikan *script* pada *button* agar bisa menjalankan perintahnya yang telah ditentukan. Dalam *software unity* menyediakan bahasa pemrograman *Javascript, C#, dan Booscrpit.* Disini penulis menggunakan bahasa pemrograman *C#* karena banyak yang digunakan oleh pengembang aplikasi saat ini.



Gambar II.1. Scripting menggunakan *software* monodevelop *unity*

> Menurut Paul Deitel, dkk (2014:47) menyimpulkan bahwa:
> Microsoft mengumumkan bahasa pemrograman C#, C#memiliki akar dalam bahasa pemrograman C, C++ dan Java. Ia memiliki kemampuan yang mirip dengan Java dan sesuai untuk tugas pengembangan aplikasi yang paling menuntut, terutama untuk membangun skala besar aplikasi perusahaan saat ini, dan, aplikasi mobile dan berbasis cloud berbasis web.

Kelebihan dari bahasa pemograman C# adalah:

1. Sederhana (simple)

   C# menghilangkan beberapa hal yang bersifat kompleks yang terdapat dalam beberapa macam bahasa pemrograman seperti *Java* dan C++, termasuk diantaranya mengilangkan *macro*, *templates*, *multiple inheritance* dan *virtual base classes*. Hal-hal tersebut yang dapat menyebabkan kebingunan pada saat menggunakannya, dan juga berpotensial dapat menjadi masalah bagi para *programmer* C++. Jika anda pertama kali belajar bahasa C# sebagai bahasa pemrograman, maka hal-hal tersebut di atas tidak akan membuat waktu anda terbuang terlalu banyak untuk mempelajarinya. C# bersifat sederhana, karena bahasa ini didasarkan kepada bahasa C dan C++. Jika anda familiar dengan

C dan C++ atau bahkan *Java*, anda akan menemukan aspek-aspek yang begitu familiar, seperti *statements, expression, operators,* dan beberapa fungsi yang diadopsi langsung dari C dan C++, tetapi dengan berbagai perbaikan yang membuat bahasanya menjadi lebih sederhana.

2. Moderen

   Apa yang membuat C# menjadi suatu bahasa pemrograman yang modern? Jawabannya adalah adanya beberapa fitur seperti *exception handling, garbage collection, extensible data types,* dan *code security* (keamanan kode/bahasa pemrograman). Dengan adanya fitur-fitur tersebut, menjadikan bahasa C# sebagai bahasa pemrograman yang modern.

3. *Object-Oriented Language*

   Kunci dari bahasa pemrograman yang bersifat *Object Oriented* adalah *encapsulation, inheritance,* dan *polymorphism. encapsulation,* dimana semua fungsi ditempatkan dalam satu paket (*single package*). *inheritance*, adalah suatu cara yang terstruktur dari suatu kode-kode pemrograman dan fungsi untuk menjadi sebuat program baru dan berbentuk suatu paket. *polymorphism,* adalah kemampuan untuk mengadaptasi apa yang diperlukan untuk dikerjakan. Sifat-sifat tersebut diatas, telah dimiliki oleh C# sehingga bahasa C# merupakan bahasa yang bersifat *Object Oriented*.

4. *Powerfull* dan *fleksibel*

   C# bias digunakan untuk membuat berbagai macam aplikasi, seperti aplikasi pengolah kata, grafik, *spreadsheets*, atau bahkan membuat kompiler untuk sebuah bahasa permrograman.

5. Efisien

C# adalah bahasa pemrograman yang menggunakan jumlah kata-kata yang tidak terlalu banyak. C# hanya berisi kata-kata yang biasa disebut dengan *keywords*. *Keywords* ini digunakan untuk menjelaskan berbagai macam informasi. Jika anda berpikiran bahwa bahasa pemrograman yang menggunakan sangat banyak kata-kata (*keywords*) akan lebih *powerfull*, maka jawabannya adalah "pemikiran itu tidak selalu benar", karena hal itu justru bisa menambah kerumitan para *developer* pada saat membuat suatu aplikasi. Berikut daftar *keywords* yang ada dalam bahasa C#

Tabel II.1. Daftar *keywords* pada bahasa C#

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	
goto	if	implicit	in	int
interface	internal	is	lock	long
namespace	new	null	object	operator
out	override	params	private	protected
public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	
static	string	struct	switch	this

throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using
virtual	void	while		

6. Modular

   Kode C# ditulis dengan pembagian masing *Class-Class* (*classes*) yang terdiri dari beberapa *routines* yang disebut sebagai member *methods. Class-Class* dan metode-metode ini dapat digunakan kembali oleh program atau aplikasi lain. Hanya dengan memberikan informasi yang dibutuhkan oleh *Class* dan metode yang dimaksud, maka kita akan dapat membuat suata kode yang dapat digunakan oleh satu atau beberapa aplikasi dan program

7. C# akan menjadi populer

   Dengan dukungan penuh dari *Microsoft* yang akan mengeluarkan produk-produk utamanya dengan dukungan *Framework.NET*, maka masa depan bahasa C# sebagai salah satu bahasa pemrograman yang ada di dalam lingkungan *Framework.NET* akan lebih baik.

Sedangkan menurut I Wayan Saryada (2004:28) mengatakan: C# adalah "suatu bahasa pemograman yang *strongly typed* ini berarti bahwa semua objek yang digunakan dalam program C# harus memiliki suatu tipe data yang spesifik dan variabel ini hanya dapat menyimpan data yang memiliki tipe yang sesuai". Misalnya kita mendeklarasikan variable bertipe int maka variable ini hanya dapat menyimpan data bilangan bulat dan tidak dapat menyimpan bilangan desimal. Selain itu kan sebagai sebuah bahasa yang *type-safe*, kompiler C# akan menjamin suatu variabel adalah tipe yang sesuai.

Variabel dalam C# dikelompokkan menjadi *variabel stastic*, *variabel instance*, variabel lokal, *elemen array*, dan parameter

variabel static adalah bagian dari tipe dan bukan merupakan bagian dari objek, oleh karena itu *variabel static* hanya dapat diakses melalui tipenya. Variabel ini juga sering disebut dengan *static field*

variabel lokal adalah variabel yang dideklarasikan didalam suatu blok, *statemen for, switch,* atau *using*. Sebelum nilai dari variabel lokal ini dapat diakses maka variabel ini perlu diberikan suatu nilai eksplisit

Berikut adalah aturan yang harus dipatuhi untuk memberikan suatu nama variabel dalam bahasa pemgrograman C#:

1. Nama variabel terdiri dari huruf, angka dan *under score* (_).
2. Nama harus diawali dengan huruf.*Under score* juga dapat digunakan untuk mengawali nama suatu variabel tetapi ini tidak disarankan.
3. C# adalah bahasa yang *case sensitif*, variabel dengan nama umur tidak sama dengan Umur.
4. *Keyword* tidak bisa digunakan sebagai nama variabel, kecuali kalau *keyword* ini diawali dengan karakter @.

Menurut Norman Sasono (2004:74) menyatakan C# adalah "sebuah bahasa pemrograman yang *object-oriented*".

Dengan menggunakan bahasa pemrograman yang *object-oriented* , anda dapat membuat sebuah program dengan *code* yang berkualitas, mudah di *maintainance* dan *code* yang dapat di *re-use* (dipakai di bagian lain dari program tanpa perlu anda meng-copy & paste baris-baris *code* tersebut).

Menulis program yang *object-oriented* sangat berbeda dengan menulis program yang bersifat *prosedural*. Untuk dapat menulis program yang *object-oriented*, anda harus memahami fitur-fitur *object-oriented* yang ada pada bahasa C#. Namun, untuk sampai ke sana, terlebih dahulu anda harus memahami dulu prinsip-prinsip dasar dari pemograman object-oriented seperti misalnya: *Abstraction, Encapsulation, Inheritance* dan *Polymorphisme*.

Menurut Fahtur Rahman (2004:89) menyimpulkan "C# adalah bahasa anda dengan komputer, dia bukan sulap. selama anda mengikuti aturan bahasanya, ia akan menuruti anda, sekali anda bikin kesalahan bahasa ia akan memaki-maki anda".

Menulis program yang *object-oriented* sangat berbeda dengan menulis program yang bersifat *prosedural*. Untuk dapat menulis program yang *object-oriented*, anda harus memahami memahami fitur-fitur *object-oriented* yang ada pada bahasa C#

prinsip-prinsip dasar dari permrograman *object-oriented* :

1. *Abstraction*
2. *Encapsulation*
3. *Inheritance*

Berikut beberapa karakter khusus pada C#:

Tabel II.2. Daftar karakter khusus pada bahasa C#

Escape Sequence	Nama	ASCII	Keterangan
\a	Bell (alert)	007	Menghasilkan suara (beep)
\b	Backspace	008	Mengembalikan posisi kursos ke sebelumnya
\t	Horizontal Tab	009	Meletakan posisi kursor di pemberhentian Tab berikutnya
\n	New line	010	Meletakan posisi kursor pada baris baru
\v	Vertical Tab	011	Melakukan Tab secara vertical
\f	Form feed	012	
\r	Carriage return	013	Memberikan nilai enter
\"	Double Quote	034	Menampilkan double quote (“)
\'	Apostrophe	039	Menamppilkan apostrophe (‘)
\?	Question mark	063	Menampilkan tanda Tanya
\\	Backslash	092	Menampilkan backslash (\)
\0	Null	000	Menampilkan karakterk Null

2.3. Metode Algorithma

a. DefinisiAlgoritma

1. Langkah - langkah yang dilakukan agar solusi masalah dapat diperoleh.

2. Suatu prosedur yang merupakan urutan langkah- langkah yg berintegrasi.

3. Suatu metode khusus yang digunakan untuk menyelesaikan suatu masalah yang nyata.

b. Sifat-Sifat Algoritma

1. Banyaknya Langkah Instruksi Harus Berhingga

2. Langkah atau Instruksi harus Jelas

3. Proses harus Jelas dan mempunyai batasan

4. Efektifitas

5. Adanya Batasan Ruang Lingkup

c. Konsep Algoritma

1. Algoritma Pe-ubah

Adalah Variabel yang nilainya bukan konstanta (selalu berubah sesuai dengan kondisi Variabel terkini)

2. Algoritma Pertukaran

Berfungsi mempertukarkan masing-masing isi Variabel sedemikian sehingga Nilai dari tiap Variabel akan berubah atau bertukar

d. Tipe Data Dalam Algoritma

1. Tipe Sederhana (simple type)

a. *Int,Bool,Char*

b. Tipe *Float*

2. Tipe *String*

a. Operasi *string*

3. Tipe Terstruktur (*structured type*)

a. *Array, Struct*

e. Tahapan Analisa Algoritma

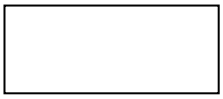
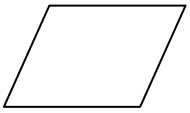
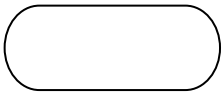
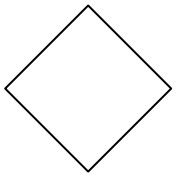
1. *Pseudocode*

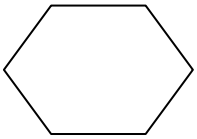
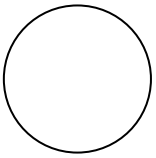
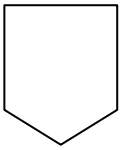
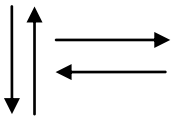
Pseudocode adalah sebuah pengembangan dari algoritma, dimana, sesuai dengan namanya, *pseudocode* menggunakan kode-kode tertentu untuk memberikan penjelasan mengenai cara kerja atau penyelesaian dari suatu masalah.

2. *Flowchart*

Flowchart adalah suatu diagram yang menggambarkan susunan logika suatu program.Simbol simbol yang digunakan dalam *flowchart* adalah sebagai berikut

Tabel II.3. Tabel simbol yang digunakan dalam *flowchart*

	Proses atau *prosessing*, satu atau beberapa himpunan penugasan yang akan dilaksanakansecara berurutan
	*Input* atau *Output* data yg akan dibaca & dimasukan ke dalam memori komputer dari suatu alat input
	Terminal, berfungsi sebagai awal (berisi '*Start*') sebagai akhir (berisi '*End*') dari suatu proses alur.
	*Decision* (kotak keputusan) berfungsi untuk memutuskan arah atau percabangan yang diambil sesuai dengan kondisi yg dipenuhi, yaitu Benar atau Salah.

	*Subroutine* digunakan untuk menjalankan proses suatu bagian (sub program) atau prosedur.
	*Preparation* digunakan untuk pemberian harga awal.
	*Connector* atau penghubung, digunakan untuk menghubungkan diagram alur yang terputus dimana bagian tersebut masih berada pada halaman yang sama.
	*On page Connector*, Untuk menghubungkan sambungan dari bagian *flowchart* yang terputus dimana sambungannya berada pada halaman dimana sambungannya berada pada halaman lain.
	*Flowline*, menunjukkan bagian arah instruksi dijalankan

f. Tahap Proses uji Algoritma

a. Fase *Debugging*

yaitu fase dari proses program eksekusi yang akan melakukan koreksi terhadap kesalahan.

b. Fase *Profilling*

yaitu fase yang akan bekerja jika program tersebut sudah benar (telah melewati *fase debugging*).

g. *Unified Modelling Language* (UML)

Unified Modelling Language (UML) adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa pemrograman berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C. Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan

di dunia. Diantaranya adalah: metodologi *booch*, metodologi *coad*, metodologi *OOSE*, metodologi *OMT*, metodologi *shlaer-mellor*, metodologi *wirfs-brock*, dsb.

Melalui berbagai penjelasan rumit yang terdapat didokumen dan buku-buku UML, sebenarnya konsepsi dasar UML dapat dirangkum seperti dibawah ini

Tabel II.4. konsep dasar UML

Major Area	**View**	**Diagrams**	**Main Concepts**
Structural	Static view	Class diagram	Class, association, generalization, dependency, realization, interface
	Use case view	Use case diagram	Use case, actor, assosiation, extend, include, use case generalization
	Implementation view	Component diagram	Component, interface, dependency, realization
	Deployment view	Deployment diagram	Node, component, dependency, location
Dynamic	State machine view	Statechart diagram	State, event, transition, action
	Activity view	Activity diagram	State, activity, completion transition, fork, join
	Interaction view	Sequence diagram	Interaction, object, message, activation
		Collaboration diagram	Collaboration, interaction, collaboration role, message
Model management	Model management view	Class diagram	Package, subsystem, model
Extensibility	All	All	Constraint, stereotype, tagged values

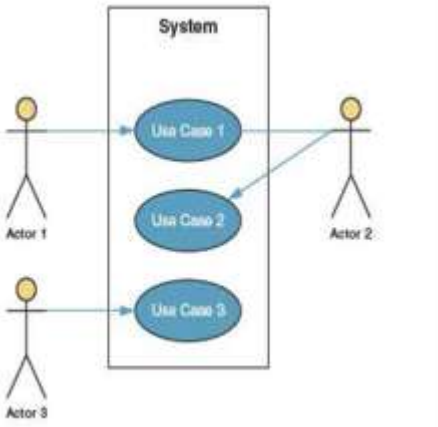
Abstraksi konsep dasar UML yang terdiri dari *structural classification, dynamic behavior*, dan model management, dapat dipahami dengan mudah apabila melihat

main concepts dapat dipandang sebagai istilah yang akan muncul pada saat membuat diagram, sedangkan *view* adalah kategori dari diagram tersebut.

Berikut merupakan beberapa diagram yang ada pada UML

1. *Use Case Diagram*

   Use case diagram adalah suatu pola atau gambaran yang menunjukan kelakuan atau kebiasan sistem berinteraksi dengan eksternal sistem dan juga user. *Use case diagram* menggambarkan siapa saja yang akan menggunakan sistem dan dengan cara apa saja yang pengguna harapkan untuk berinteraksi dengan sistem (Whitten dan Bentley, 2007).



Gambar II.2. Contoh *Use Case Diagram*

a. *Use Case*

   Mempresentasikan gambaran fungsional dari sistem yang setiap langkahnya berurutan (skenario). *Use case* digambarkan dalam bentuk *elips* dan nama dari *use case* ditulis di atas, bawah maupun di dalam *elips* (Whitten dan Bentley, 2007).

Gambar II.3. Contoh Use Case

b. Actor

Mempresentasikan sesuatu (perangkat, sistem lain) yang berinteraksi dengan sistem untuk pertukaran informasi (Whitten dan Bentley, 2007).

Gambar II.4. Contoh Aktor

2. *Class Diagram*

Class diagram adalah menggambarkan struktur objek system yang menunjukan kelas objek dari system tersebut bahwa sistem ini terdiri dari hubungan antara kelas-kelas objek (Whitten dan Bentley, 2007).

Sebuah *class diagram* disusun oleh tiga bagian, yaitu:

a. Nama Kelas

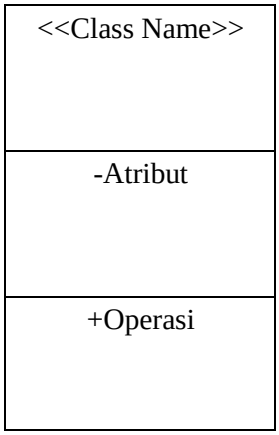
Merupakan nama dari suatu *class* yang digunakan sebagai pembeda dari suatu objek.

b. Atribut Kelas

Merupakan atribut atau data yang berada pada *class* tersebut.

c. Operasi Kelas

Merupakan fungsi atau operasi yang dimiliki pada *class* tersebut.



Gambar II.5. Struktur dari *Class Diagram*

3. *Sequence Diagram*

Sequence diagram adalah grafis yang menggambarkan bagaimana objek berinteraksi satu sama lain melalui pesan dalam pelaksanaan use case atau pengoperasian (Whitten dan Bentley, 2007).

Berikut komponen dari sequence diagram:

a. *Actor*

Actor yang berinteraksi dengan *interface*.

Gambar II.6. Contoh *Actor*

b. *Lifeline*

Lifeline bertujuan untuk menunjukan kehidupan dari sebuah *sequence.*

Gambar II.7. Contoh *Lifeline*

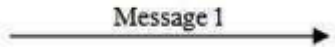
c. *Activation*

Bar yang berada di atas *lifelines* yang menunjukan periode waktu dimana sistem aktif dalam interaksi.

Gambar II.8. Contoh *Activation Bar*

d. *Message*

Message menunjukan pesan yang masuk dikirim ke dalam *class*.



Gambar II.9. Contoh *Message*

e. *Message* (*Return*)

Return message menunjukan setiap kegiatan yang kembali, pesan benar atau salah yang menunjukan apakah perilaku itu berhasil.

Gambar II.10. Contoh *Return Message*

f. *Self-Call*

Sebuah objek yang dapat memanggil methodnya sendiri.

Gambar II.11. Contoh *Self-Call*

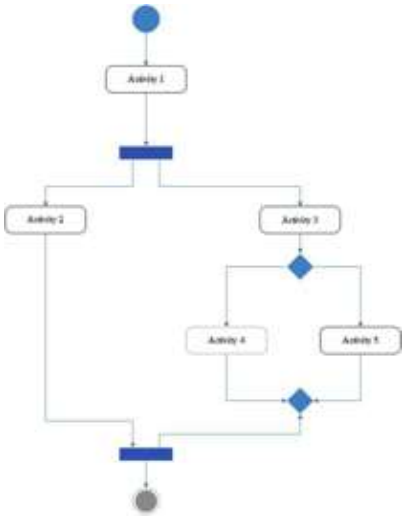
g. *Object*

Gambar yang mewakili kelas-kelas yang berasal pada *class diagram*.

Gambar II.12. Contoh *Object*

4. *Activity Diagram*

Activity diagram adalah diagram yang menggambarkan kegiatan yang sekuensial dari *use case* atau proses bisnis, dan juga dapat digunakan untuk model logika sistem (Whitten dan Bentley, 2007).



Gambar II.13. Contoh *Activity Diagram*

Berikut ini merupakan notasi-notasi yang digunakan dalam *activity diagram*

a. *Initial State*

Menampilkan awal dari proses.

Gambar II.14. Contoh dari *Initial State*

b. *Action*

Menjelaskan individual *steps.*

Gambar II.15. Contoh dari *Action*

c. *Flow*

Menjelaskan progress dari sebuah *action*.

Gambar II.16. Contoh dari *Flow*

d. *Decision* atau *Merge*

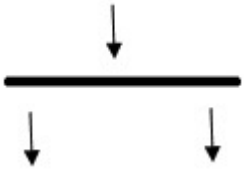
Decision berbentuk *diamond* dengan satu arus masuk dan dua atau lebih arus keluar. Arus yang keluar ditandai untuk mengindikasikan kondisi. *Merge* berbentuk *diamond* dengan dua atau lebih arus masuk dan satu arus yang keluar.Ini menggabungkan arus yang sebelumnya dipisahkan oleh *decisions*. Proses berlanjut dengan adanya salah satu arus yang masuk ke dalam *merge*.

Gambar II.17. Contoh dari *Decision* dan *Merge*

e. *Fork* atau *Join*

Fork merupakan balok hitam dengan satu arus yang masuk dan dua atau lebih yang keluar. Tindakan pada arus paralel dapat terjadi dalam urutan apapun ataupun bersamaan.

Join merupakan balok hitam dengan dua atau lebih arus yang masuk dan satu arus yang keluar, Semua aksi yang masuk harus selesai sebelum proses lainnya dilanjutkan.



Gambar II.18. Contoh dari *Fork* dan *Join*

f. *Final State*

Menjelaskan akhir dari proses.

Gambar II.19. Contoh dari *Final State*

2.4. Pengujian Sistem

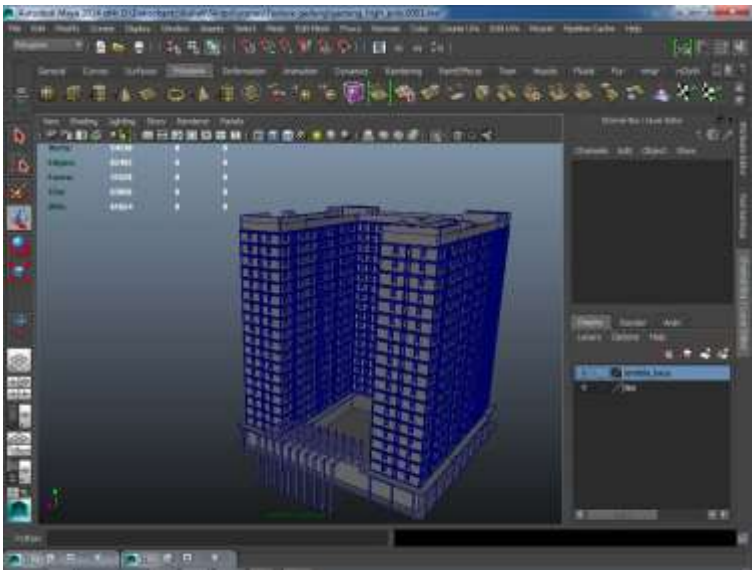
White box testing merupakan jenis testing yang lebih berkonsentrasi terhadap isi dari perangkat lunak itu sendiri. Jenis ini lebih banyak berkonsentrasi kepada *source code* dari perangkat lunak yang dibuat sehingga membutuhkan proses testing yang jauh lebih lama dan lebih mahal dikarenakan membutuhkan ketelitian dari pada *tester* serta kemampuan teknis pemrograman bagi para testernya

Black box adalah tipe testing yang memperlakukan perangkat lunak yang tidak diketahui kinerja internalnya. Sehingga para *tester* memandang perangkat lunak seperti layaknya sebuah kotak hitam yang tidak penting dilihat dari isinya, tapi cukup dikenai proses testing dibagian luar

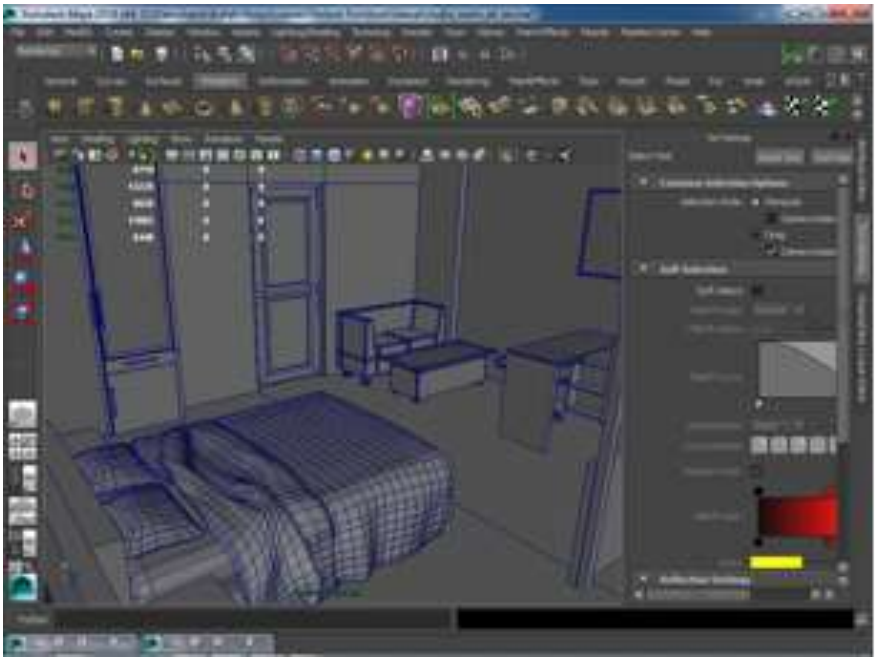
2.5. Peralatan Pendukung

1. Autodesk Maya

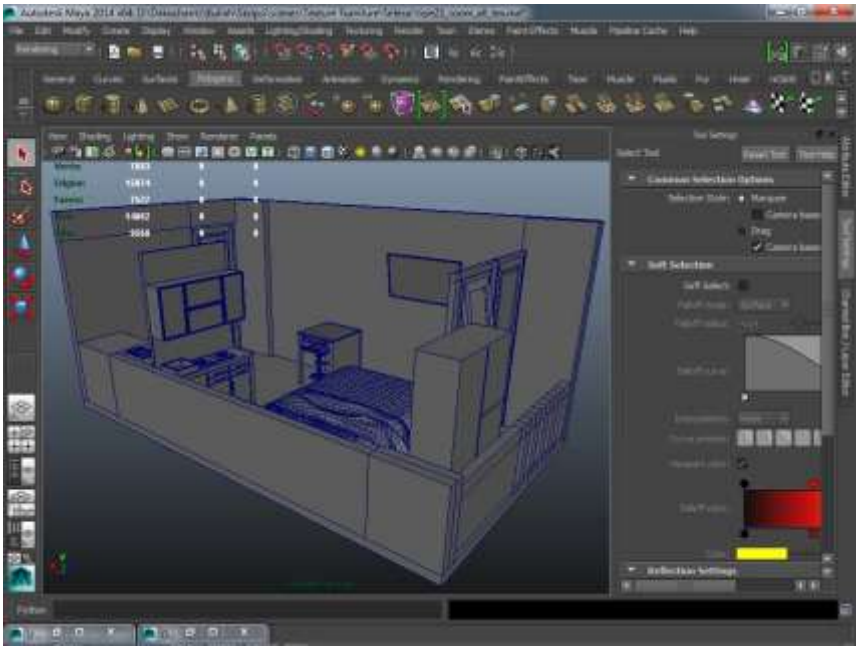
Tahap pertama penulis menggunakan *software autodesk maya* untuk membuat pemodelan 3D visual gedung serta tipe ruangan apartemen.



Gambar II.20. *Modelling* 3D gedung apartemen menggunakan *software autodesk maya*

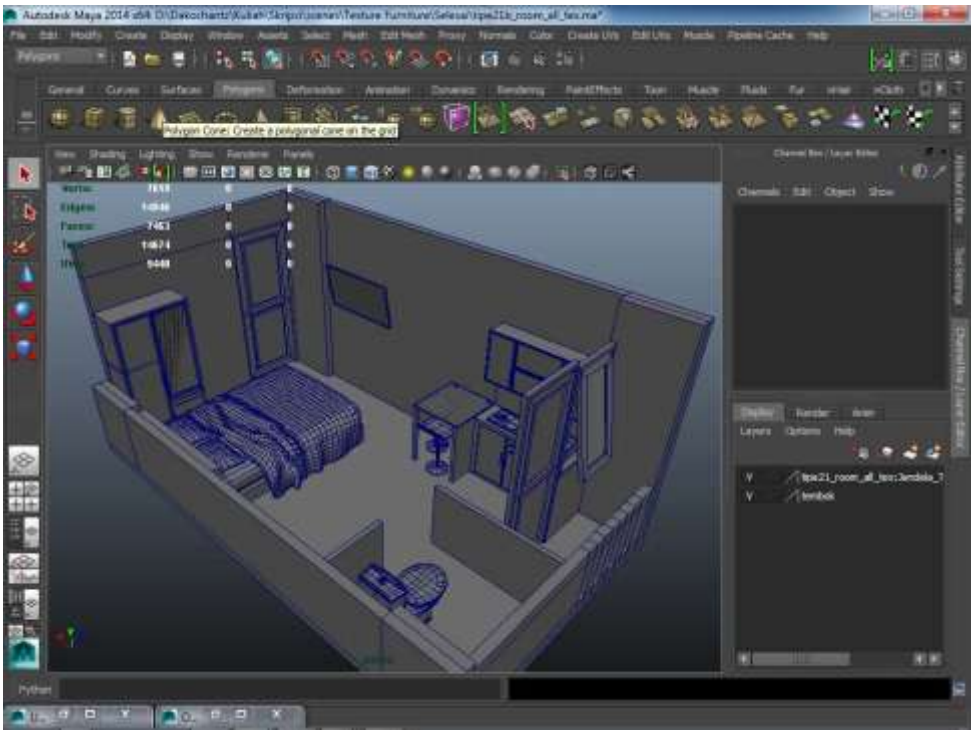


Gambar II.21. ***Modelling*** **3D interior menggunakan** ***software autodesk maya***

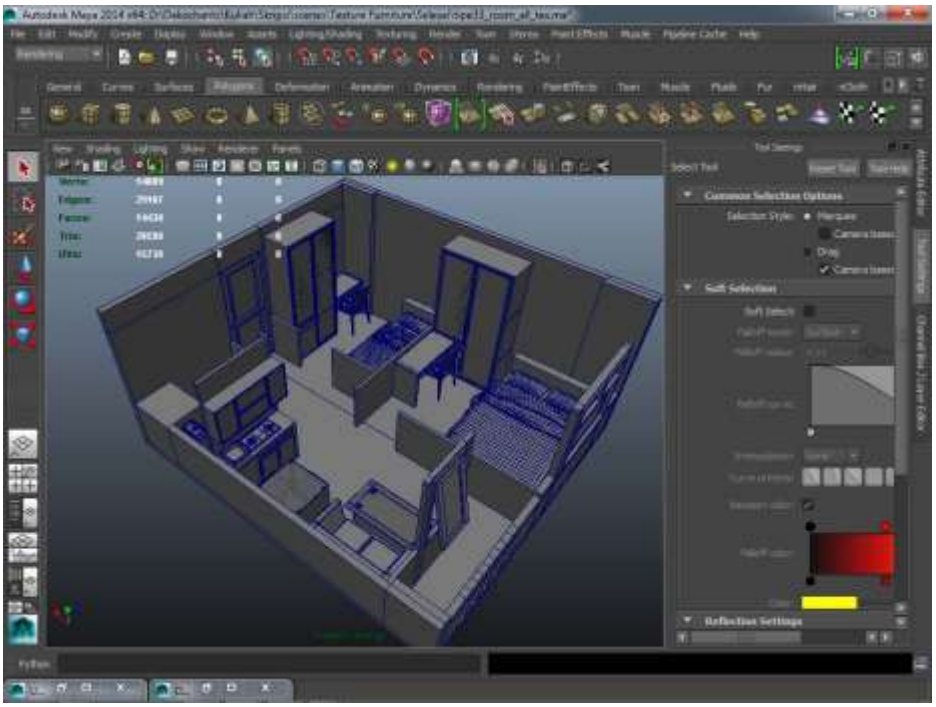


Gambar II.22. ***Modelling*** **3D ruangan tipe 21 menggunakan** ***software autodesk maya***

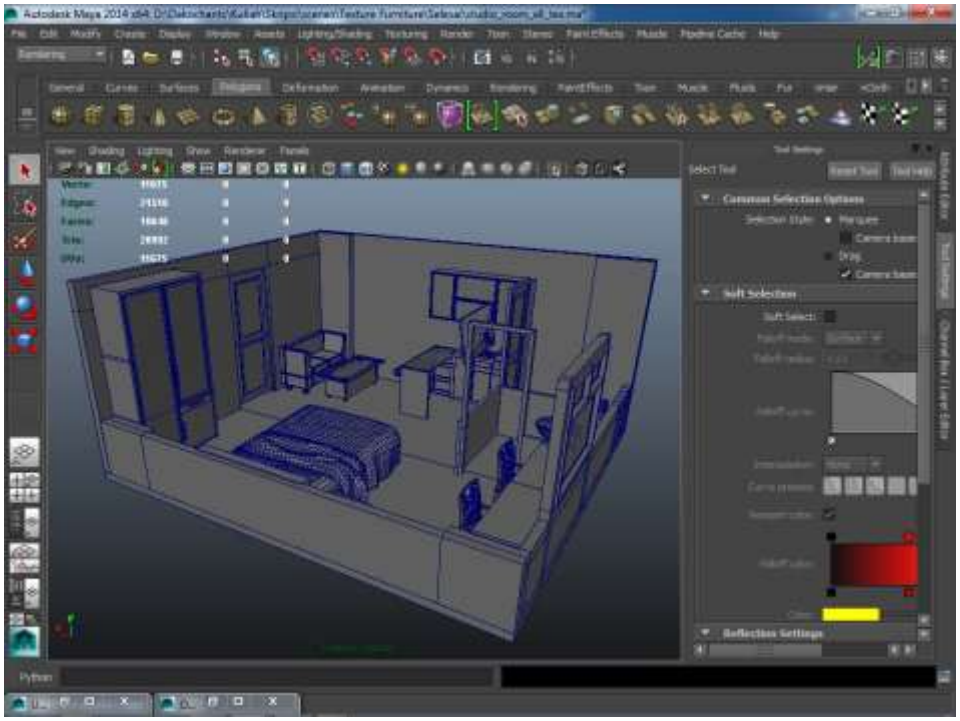
.



Gambar II.23. ***Modelling*** **3D ruangan tipe 21studio menggunakan** ***software autodesk maya***



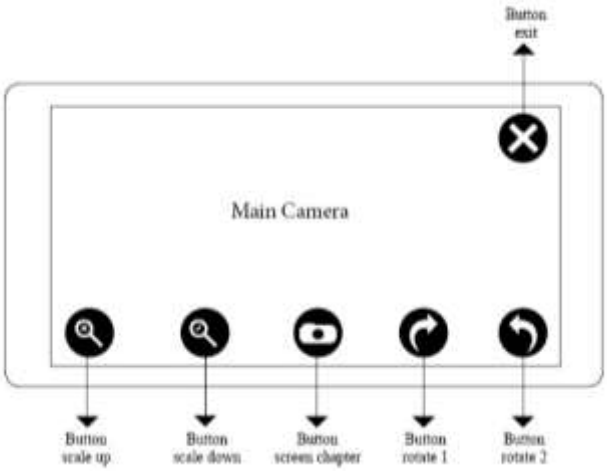
Gambar II.24. ***Modelling*** **3D ruangan tipe 33 2** ***bed room*** **menggunakan** ***software autodesk maya.***



Gambar II.25. ***Modelling*** **3D ruangan tipe 33 studio menggunakan** ***software autodesk maya.***

2. Adobe Illustrator

Tahap pertama penulis menggunakan *software adobe illustrator* untuk membuat ilustrasi *infographic*, *Layout* serta tipografi untuk media pendukung aplikasi bergerak media promosi apartemen

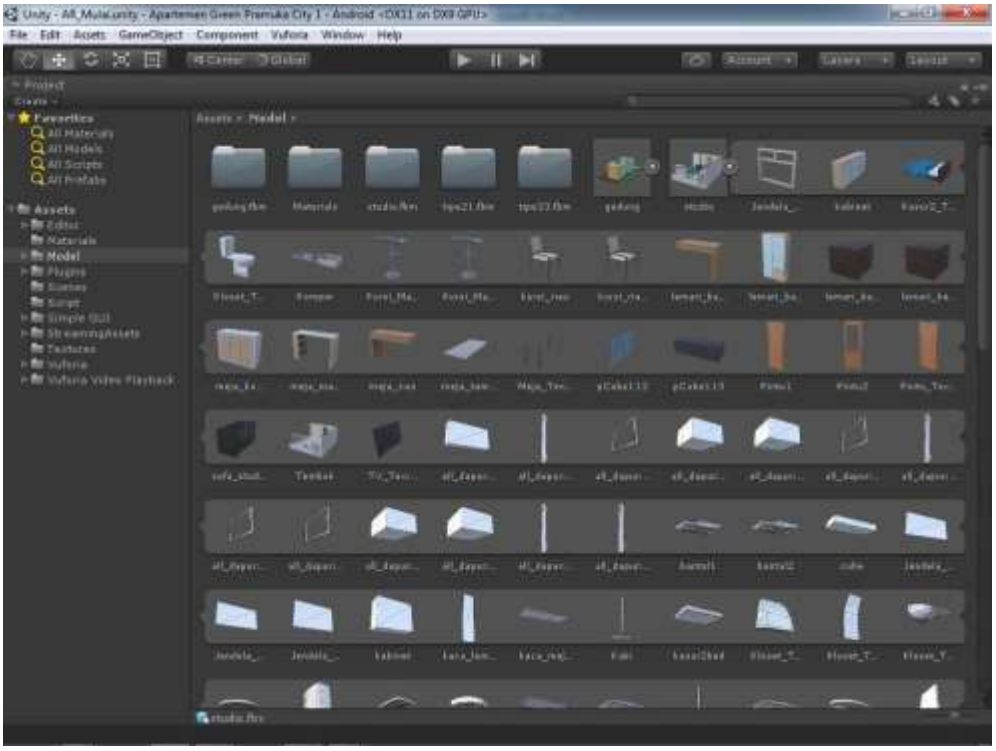


Gambar II.26. ***Tracing infographic*** **menggunakan** ***software adobe illustrator***

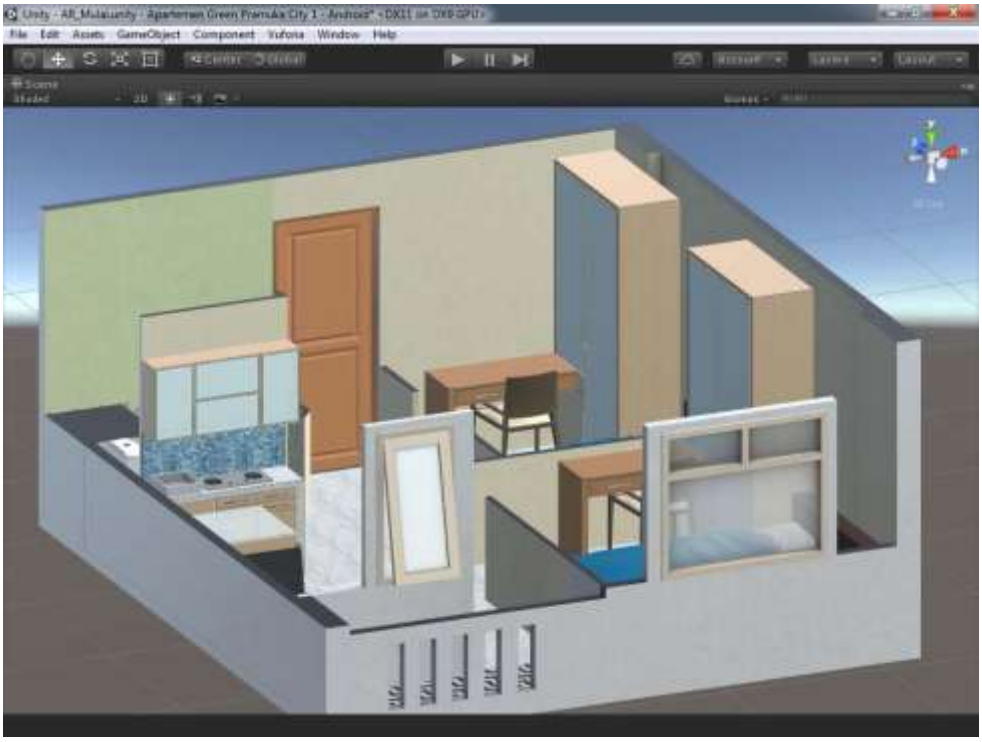
Gambar II.27. *Layouting* ilustrasi *interface* aplikasi menggunakan *software adobe illustrator*

3. Unity 3D

Tahap ini adalah penggabungan objek tiga dimensi dengan *layout interface* yang telah dibuat sebelumnya menggunakan *software* masing-masing dan telah di *export* dengan format yang bisa terbaca pleh *software compositig*, disini penulis menggunakan *software developer unity* untuk membangun aplikasi bergerak media promosi apartemen menggunakan teknologi *augmented reality* dan berbasis sistem operasi android.



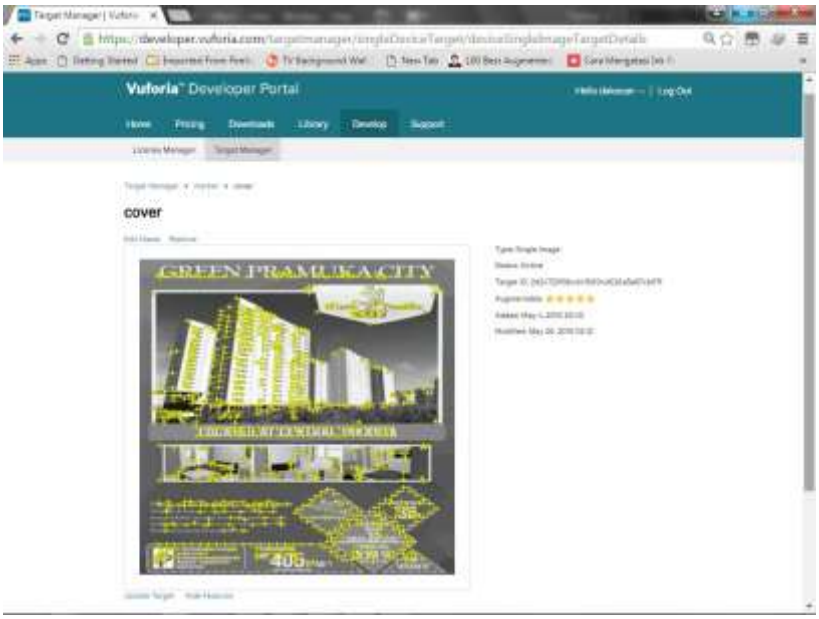
Gambar II.28. *Import* semua *modelling* 3D yang telah menggunakan *software unity.*



Gambar II.29. Compositing *modelling* 3D hingga menjadi ruangan yang utuh dengan menggunakan *software unity.*

4. Markerless

Tahap ini adalah pembuatan *database* menggunakan aplikasi *Qualcomm Vuforia Developer*. Proses ini membuat *database* untuk *marker* yang ingin digunakan agar bisa terbaca oleh aplikasi dengan cara *mengupload* file *image* yang telah dibuat kedalam *server Qualcomm Vuforia Developer*, setelah selesai *upload server Qualcomm* akan memproses *image* yang telah di*upload* menjadi *marker*. Selanjutnya *marker* yang telah selesai diproses di-*download* dengan format *unity editor* atau *unitypackage*. Proses ini dilakukan agar *database marker* dapat terbaca dan di-*import* pada *software compositing*, yaitu *software unity* untuk di-*build* bersama dengan aplikasi yang telah dibuat menjadi file. APK adalah format ekstensi file *installer* untuk sistem operasi android. Inilah tahapan bagaimana palikasi bisa membaca *marker* yang digunakan untuk diproses menjadi visual objek tiga dimensi.



Gambar II.30. *Markerless* menggunakan *Developer Vuforia*