BAB II

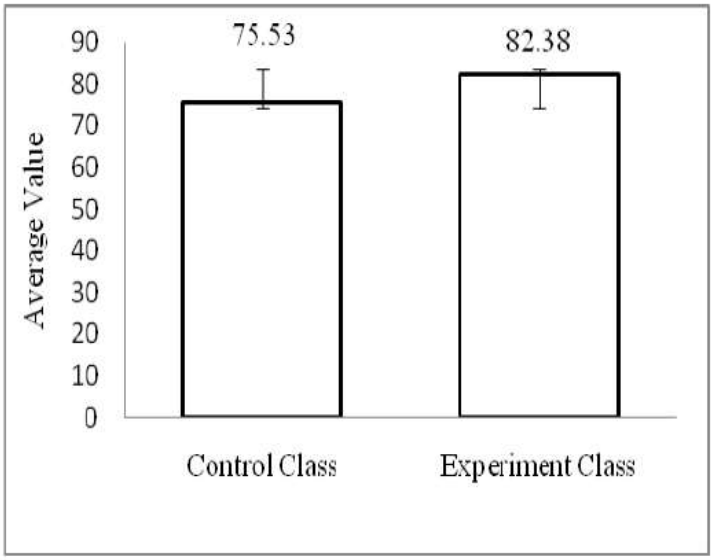
TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Beberapa penelitian terdahulu, telah membahas mengenai penggunaan *virtual lab* sebagai media pembelajaran untuk meningkatkan efektivitas pembelajaran dibandingkan tanpa menggunakan *virtual lab*. Penelitian-penelitian ini disadur ringkas dalam bagian-bagian pada subbab ini.

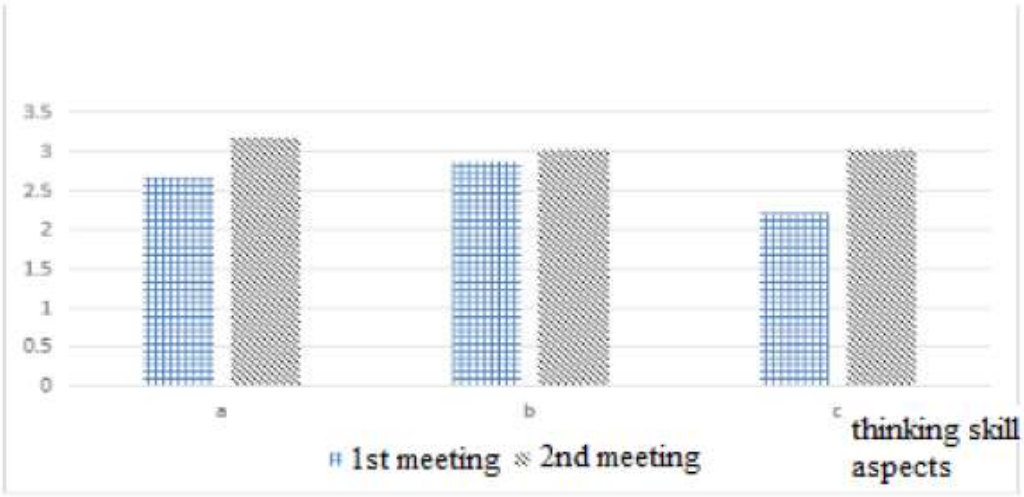
Penelitian pertama, Dalam penelitian oleh Syahfitri (2019:64), peneliti tersebut mencoba mengetahui kelayakan dari sebuah *virtual lab* berbasis *problem*, dalam subtopik ekosistem. Dalam penelitian ini, penulis mencoba mengetahui pendapat pakar terhadap kelayakan sebuah *virtual lab* dari sisi pakar, kelayakan dilihat dari sisi respon pelajar dan guru, dan efektivitas dari penggunaan media *virtual lab* dalam mencapai tujuan pembelajaran. *Virtual lab* yang dirancang dalam penelitian ini, menggunakan pendekatan berbasis permasalahan (*problem based*) yang berarti *virtual lab* ini mengemukakan sebuah permasalahan untuk dipecahkan oleh pengguna akhir. Dari pseudo-eksperimen yang dilakukan oleh peneliti, berhasil diperoleh tingkat kelayakan penggunaan *virtual lab* oleh pakar dan pengguna di atas 79%. Selain itu sebagaimana ditunjukkan pada Gambar 2.1, didapati rata-rata nilai *post-test* dari kelompok siswa yang menggunakan *virtual lab* lebih tinggi dibandingkan yang kelompok siswa yang tidak menggunakan *virtual lab*, yaitu 82.38 dibandingkan 75.53 dari nilai maksimum 100.

Disimpulkan bahwa penggunaan *virtual lab* mampu mendukung efektivitas proses pembelajaran.



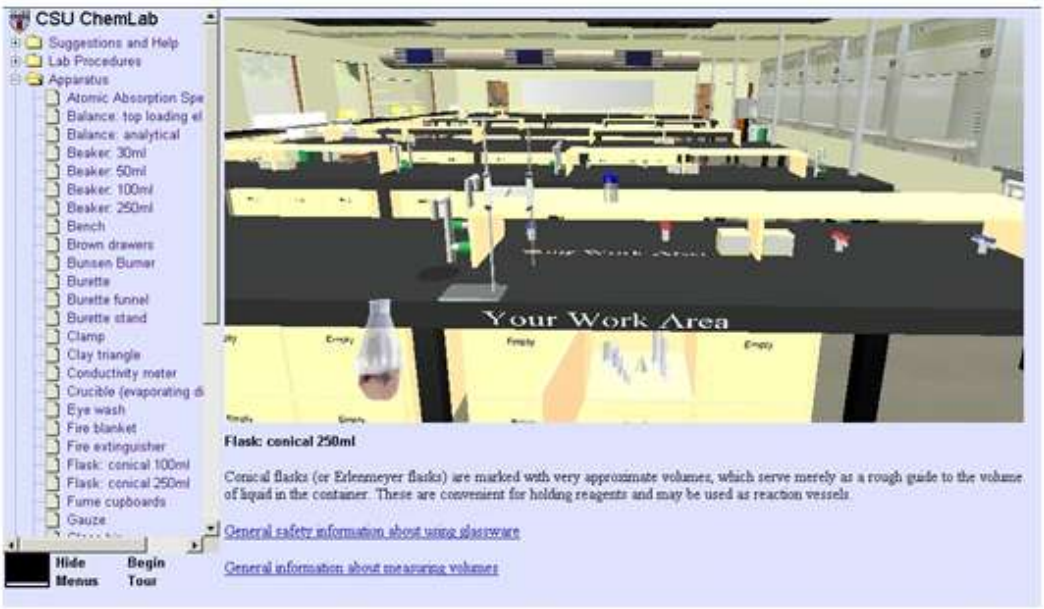
Gambar 2.1 Grafik Nilai Rata-Rata Post Test

Penelitian kedua, Penelitian dan pengembangan oleh Widowati (2017:170) meneliti mengenai pengembangan *virtual lab* berbasis *inquiry*. Pengembangan ini menggunakan pengembangan dalam empat tahap (4D) yaitu *define*, *design*, *develop*, *disseminate*. Penelitian ini menemukan bahwa penggunaan *virtual lab* untuk menunjang pembelajaran, meningkatkan efektivitas pembelajaran dibandingkan apabila tanpa menggunakan *virtual lab*. Namun, penelitian ini juga menemukan bahwa kedudukan *virtual lab* masih belum mampu menggantikan kemampuan dari laboratorium nyata (fisik) dalam menunjang proses pembelajaran didalam kegiatan pendidikan. Efektivitas penggunaan ditunjukkan pada Gambar 2.2.



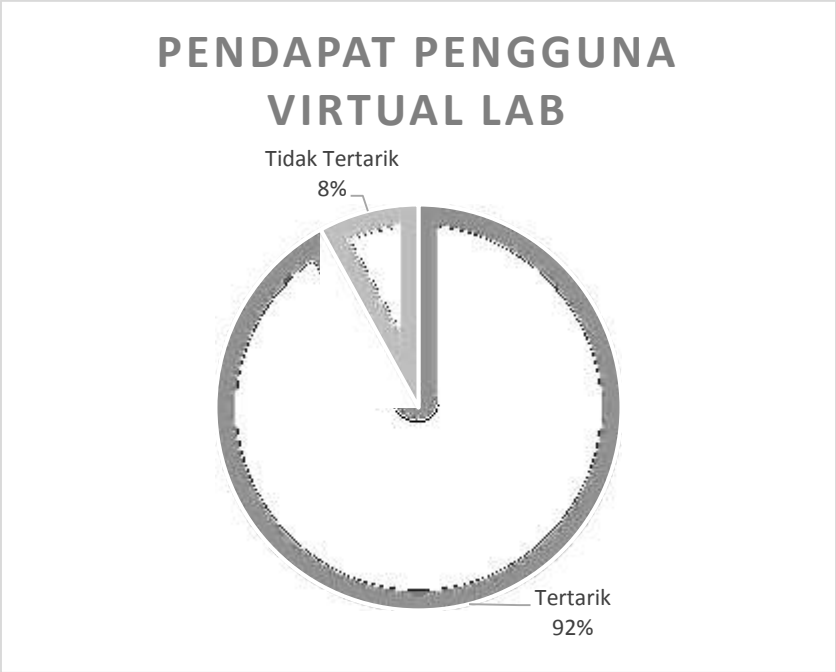
Gambar 2.2 Hasil Pengujian *Virtual Lab* Widowati.

Penelitian ketiga, Penelitian lain oleh Dalgarno (2009:853) membahas mengenai penggunaan media *virtual lab* dalam eksperimen kimia, seperti pada Gambar 2.3. Penelitian yang dilakukan menunjukkan bahwa beberapa kasus penggunaan *virtual lab* untuk pembelajaran kimia dan sains, terlepas dari kemampuannya untuk menjadikan pelajar familiar dengan penggunaan laboratorium, bukanlah penentu utama keberhasilan penggunaan *virtual lab* dalam pembelajaran. Namun, penentu keberhasilan dalam suasana *virtual lab* adalah kemampuan dari pelajar untuk menerapkan konsep teoritis dan matematis, dalam hubungannya untuk penyelesaian permasalahan yang dihadapi. Sehingga rekomendasi dari penelitian ini adalah untuk penyediaan sarana atau peralatan pelengkap yang dapat membantu pelajar untuk menarik hubungan antara teori yang telah dipelajari, dengan penggunaannya, terutama pada situasi pemecahan masalah.

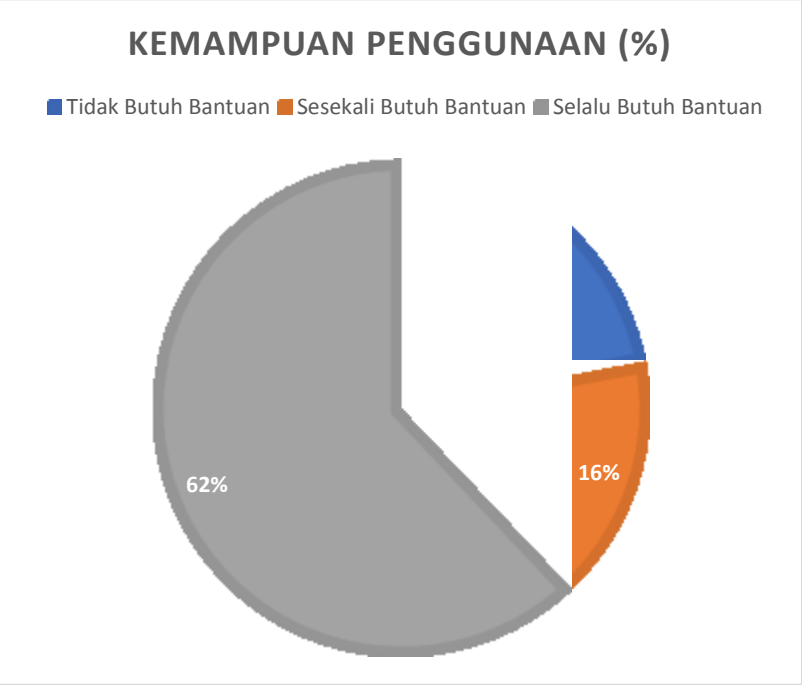


Gambar 2.3 Contoh Aplikasi Virtual Lab oleh Dalgarno

Penelitian keempat, Penelitian oleh Rajendran (2010:2173) dalam pengembangan *virtual lab* membahas mengenai penggunaan *virtual lab* di kalangan sekolah menengah. Penelitian ini lebih membahas pada respon dan kemampuan *virtual lab* dalam meningkatkan pemahaman oleh peserta pembelajaran dalam menerima materi atau topik yang diajarkan. Penelitian menghasilkan kesimpulan yang mendukung hasil-hasil penelitian sebelumnya yang mendukung penerapan *virtual lab* dalam kaitannya untuk menunjang pembelajaran.

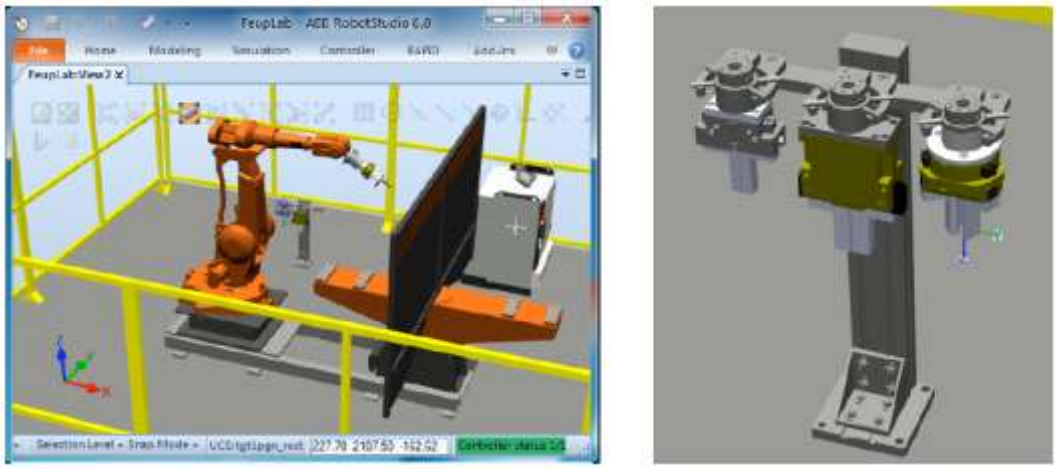


Gambar 2.4 Respon Pengguna Terhadap *Virtual Lab*



Gambar 2.5 Kemampuan Pengguna Mengoperasikan *Virtual Lab*

Penelitian kelima, Penelitian oleh Abreu (2015:10) mengusulkan mengenai penggunaan *virtual lab* dalam topik pemrograman untuk robot industrial. Pada penelitian ini ditunjukkan mengenai proses pengembangan, topik dan cakupan yang dibahas dalam *virtual lab*, hingga perangkat lunak penunjang pengembangan *virtual lab* ini. Kesimpulan yang didapatkan dalam penelitian ini adalah usulan komponen yang dibutuhkan untuk mengembangkan suatu *virtual lab* yang efektif dalam mendukung pembelajaran.



Gambar 2.6 Aplikasi Virtual Lab Robotik Industrial

2.1.1 Konsep *Waterfall Model* Oleh Benington

Dalam konsep yang diusulkan oleh Benington (1983:350) dikemukakan sebuah konsep pemgembangan perangkat lunak dengan basis *waterfall*. Pendekatan *waterfall* dalam pengembangan perangkat lunak, merupakan pendekatan pengembangan yang menjabarkan kebutuhan, dan perancangan pada bagian awal proyek, dan mendedikasikan sebagaian besar sumberdaya proyek pada bagian perencanaan. Pada pendekatan ini, perubahan terhadap definisi yang

telah dikembangkan menjadi lebih terbatas, karena asumsi bahwa ruang lingkup pekerjaan dan kebutuhan sudah ditetapkan dan cenderung tidak berubah hingga akhir periode pengerjaan proyek. Pendekatan ini akan dapat mengidentifikasi permasalahan yang mungkin timbul, di awal pengerjaan, sehingga dapat ditemukan solusi terhadap sebagian besar permasalahan yang ada. Namun kritik yang timbul atas pendekatan ini adalah bahwa pada ranah pengembangan perangkat lunak, pendekatan ini seringkali kurang cocok, karena beberapa aspek pendekatan perangkat lunak cenderung tidak diketahui pada awal pelaksanaan proyek, dan kemungkinan akan mengalami perubahan di masa depan. Terlepas dari itu, apabila bahwa kebutuhan pengembangan telah dibatasi, sebagaimana pada proyek tugas akhir ini, maka pendekatan ini menjadi lebih cocok untuk diterapkan, karena minimnya perubahan yang akan terjadi.

2.2 Teori Terkait

Pada subbab ini, akan dibahas secara ringkas, dasar teori yang terkait dengan pengembangan *virtual lab* yang akan dilakukan, termasuk perangkat lunak yang digunakan. Beberapa hal yang akan dibahas antara lain adalah IDE, Gamemaker Studio 2, Library, Extensions, OOP, dan komponen perangkat keras *personal computer* yang akan dibahas dalam *virtual lab*.

2.2.1 *Integrated Development Environment* (IDE)

Sebuah *Integrated Development Environment* (IDE) adalah sebuah aplikasi perangkat lunak yang menyediakan fasilitas yang komprehensif, kepada programmer untuk pengembangan perangkat lunak. Sebuah IDE umumnya terdiri setidaknya dari *editor* untuk melakukan penyuntingan terhadap *source code*,

kemudian perangkat untuk melakukan otomasi *build*, dan umumnya perangkat untuk melakukan *debugging*. Namun, kemampuan IDE tidak terbatas pada hal-hal ini saja, karena beberapa IDE semisal NetBeans dan Eclipse, memiliki perangkat untuk melakukan *compile* ataupun interpreter yang diperlukan.

Namun, dalam pelaksanaannya, tidak terdapat batasan yang jelas antara IDE dengan bagian-bagian lain dari perangkat pengembangan perangkat lunak yang lainnya. Beberapa IDE mendukung penggunaan *version control system*, dan bermacam perangkat lain untuk menyederhanakan pembuatan unsur-unsur GUI. Hampir seluruh IDE modern memiliki peramban untuk kelas dan obyek, dalam lingkungan pengembangan perangkat lunak berbasis obyek.

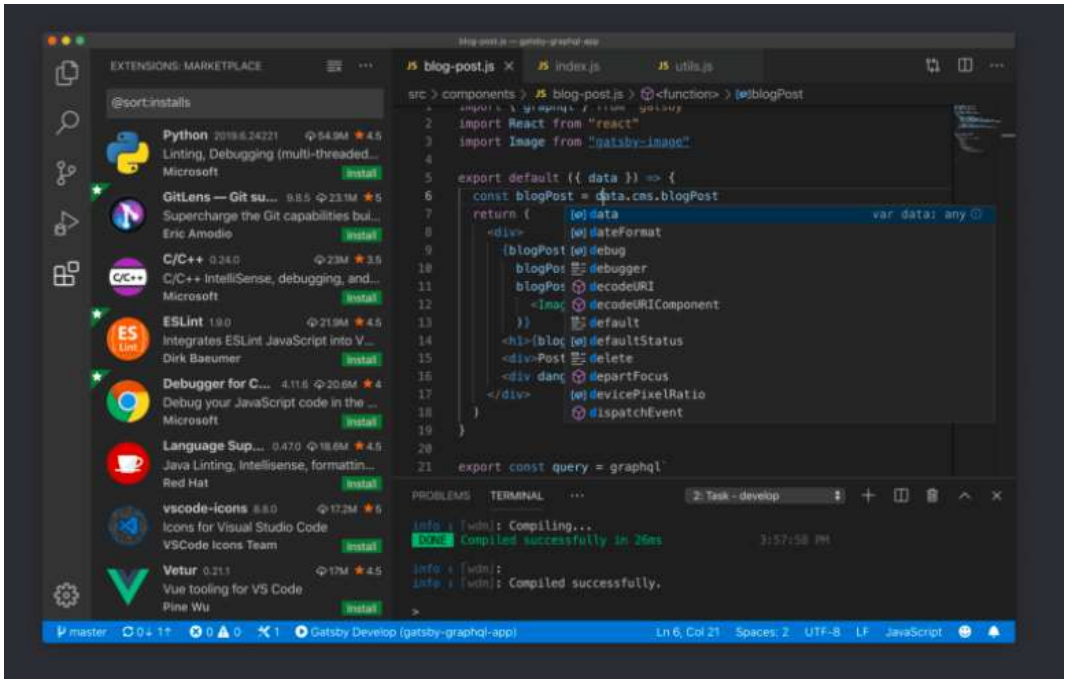
Rancangan dari IDE sendiri memfokuskan untuk dapat memaksimalisasi produktivitas dari programmer, dengan menyedikan komponen-komponen yang berhubungan dengan menggunakan antar muka yang memiliki kemiripan. IDE menampilkan sebuah program dimana seluruh kegiatan pengembangan dilakukan. Program-program ini umumnya menyediakan bermacam-macam fitur untuk menulis, memodifikasi, mengkompilasi, *deployment*, dan *debugging* perangkat lunak. Hal ini berbeda jauh dengan pengembangan perangkat lunak yang menggunakan berbagai macam peralatan pengembangan yang tidak berhubungan satu sama lain.

Selain itu, tujuan dari penggunaan IDE adalah untuk mengurangi konfigurasi yang diperlukan untuk menyusun berbagai macam perangkat pembantu pengembangan, tetapi dengan menyediakan satu set kemampuan pengembangan sebagai sebuah unit yang kohesif. Dengan berkurangnya

kebutuhan waktu untuk melakukan setup terhadap pengembangan, secara umum akan meningkatkan produktivitas dari pengembang aplikasi, terutama pada kasus-kasus dimana mempelajari sebuah IDE akan lebih cepat dibandingkan integrasi masing-masing perangkat pengembangan individu secara terpisah.

Integrasi menyeluruh terhadap semua kegiatan pengembangan perangkat lunak ini berimbas terhadap peningkatan produktivitas secara umum, diluar kegiatan terkait setup. Sebagai contoh, kode dapat di-*parse* ketika sedang disunting, sehingga dapat langsung memberikan umpan balik terhadap programmer apabila terjadi kesalahan dari *syntax* yang ditulis, sehingga dapat mencegah kesalahan dan *bug* untuk muncul belakangan.

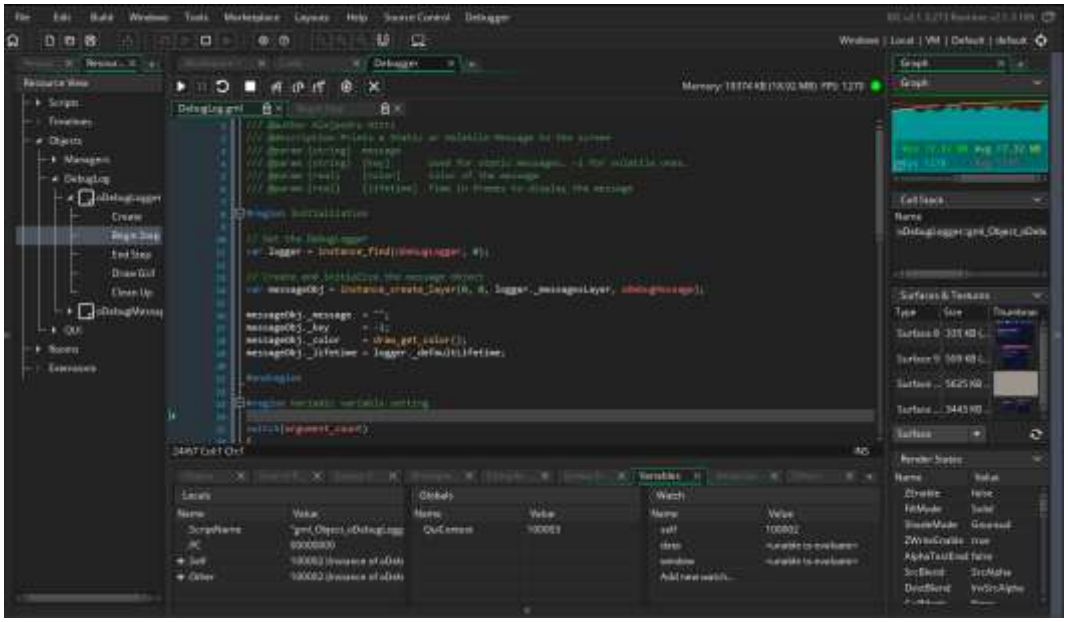
Beberapa IDE secara khusus didedikasikan untuk bahasa pemrograman yang spesifik, yang memungkinkan penggunaan beberapa fitur yang sangat terkait erat dengan paradigma pemrograman dari bahasa pemrograman tersebut. Namun, banyak juga terdapa IDE yang mendukung berbagai bahasa pemrograman, salah satunya adalah Visual Studio Code yang ditunjukkan pada Gambar 2.1.



Gambar 2.7 Contoh IDE Visual Studio Code

2.2.2 Gamemaker Studio 2

Gamemaker Studio 2, awalnya bernama Animo, dan berubah menjadi Game Maker merupakan sebuah game engine lintas *platform* yang mulai dikembangkan oleh Mark Overmars pada 1999. Pengembangan selanjutnya dilakukan oleh YoYo Games sejak 2007, dengan iterasi terbaru yaitu Gamemaker Studio 2 yang dirilis pada 2017. Gambar 2.2 menunjukkan tampilan Gamemaker Studio 2.



Gambar 2.8 Gamemaker Studio 2 IDE

Gamemaker Studio 2 (selanjutnya akan disebut GMS2) mengakomodasi pengembangan lintas *platform* dan *multi-genre* untuk video *games* dengan menggunakan dua dukungan utama: DnD *Visual Programming*, dan bahasa pemrograman GML. Dari sisi kelengkapan, GML memiliki akses dan kapabilitas lebih baik dibandingkan DnD, sehingga kebanyakan pengembangan oleh pengembang perangkat lunak menggunakan bahasa GML. Pada awalnya GMS2 berfokus untuk memfasilitasi programmer pemula dalam melakukan pengembangan game, namun iterasi terakhir dari GMS2 lebih banyak berfokus untuk menyasar pengembang yang telah memiliki pengalaman.

Fokus utama dari IDE gamemaker adalah untuk pengembangan *game* dan aplikasi dengan grafis 2D, memungkinkan penggunaan langsung grafik raster, grafik vektor, dan animasi skeletal 2D, bersamaan dengan *library standard* yang cukup besar untuk proses penggambaran grafik 2D secara primitif. Walaupun IDE ini memiliki kemampuan untuk grafik 3D, penggunaannya dilakukan dengan

menggunakan *vertex buffer* dan fungsi-fungsi kalkulasi *matrix*, sehingga tidak terlalu mudah diakses oleh pengguna pemula.

Selain itu, *engine* Gamemaker menggunakan Direct3D pada *platform* Windows, UWP, dan Xbox One sebagai API untuk grafik. Pada macOS dan linux, digunakan OpenGL, dan OpenGL ES untuk android dan iOS, serta WebGL dan 2dCanvas pada HTML5. Salah satu kelebihan yang dimiliki oleh GMS2 adalah *built-in editor* untuk penyuntingan grafik raster, desain level, *scripting*, *pathing*, dan *shaders*. Fungsionalitas lainnya dapat diimplementasikan dengan menggunakan bahasa pemrogramannya, atau dengan menggunakan ekstensi *native* untuk *platform* sasaran.

Bahasa pemrograman yang digunakannya, GML, merupakan sebuah bahasa pemrograman yang serupa dengan JavaScript dan C, namun ditulis secara dinamis dan *imperative*. Bahasa ini juga dapat dikompilasi secara *source to source* dengan menggunakan YYC untuk mendapatkan kinerja yang lebih baik. Pada ekspor untuk *platform* HTML5, gml dikompilasi secara *source-to-source* menjadi Javascript, dengan optimasi dan minifikasi diterapkan pada *build non-debug*.

2.2.3 Library

Pada bidang rekayasa informatika dan ilmu komputer, sebuah *library* merupakan kumpulan dari sumberdaya *non-volatile* yang digunakan oleh program, umumnya untuk pengembangan perangkat lunak. Sumberdaya ini mungkin berisi data konfigurasi, dokumentasi, data dukungan, *template*, kode, fungsi dan *subroutine* yang telah ditulis sebelumnya, kelas, nilai, dan spesifikasi dari tipe data. Selain itu, *library* juga merupakan kumpulan implementasi suatu

pola sikap yang ditulis berdasarkan suatu bahasa pemrograman, yang memiiki antarmuka dengan tingkat kecocokan tinggi terhadap pola sikap yang dipanggil. Sebagai contoh, programmer yang menulis program menggunakan bahasa tingkat tinggi hanya perlu memanggil *library* untuk menggunakan suatu fungsi atau *subroutine* berkali-kali, tanpa perlu menulis ulang setiap kali program tersebut diimplementasikan.

Selain itu, sikap dari sistem seperti ini menampilkan kemungkinan penggunaan suatu fungsi atau *subroutine* secara berulang-ulang dari berbagai program terpisah yang tidak berkaitan. Program dapat memanggil suatu sikap program yang diinginkan dari *library* yang digunakan melalui mekanisme yang tersedia dalam bahasa tersebut. Sebagai contoh implementasi ini adalah *tensorflow* yang memiliki *library* pada bahasa yang berbeda, yang mana menghasilkan implementasi yang sama pada tingkat mesin. Pada sistem seperti yang menggunakan bahasa C, *library* ini dipanggil menggunakan *function-call* biasa, dan tidak ada perbedaan yang signifikan, selain pola organisasi kode didalam sistem.

Library juga diorganisir sedemikian sehingga dapat digunakan oleh berbagai program yang tidak berhubungan satu sama lain, sementara kode yang terkait untuk program tersebut, digunakan untuk program tersebut. Perbedaan ini dapat menghasilkan nosi yang besar, ketika digunkan pada program-program besar yang mungkin memiliki jutaan baris kode pemrograman. Pada perbedaannya yang mendasar, *library* dibuat dengan tujuan untuk penggunaan ulang pada program lain, yang mana hal ini akan berbeda dengan suatu fungsi

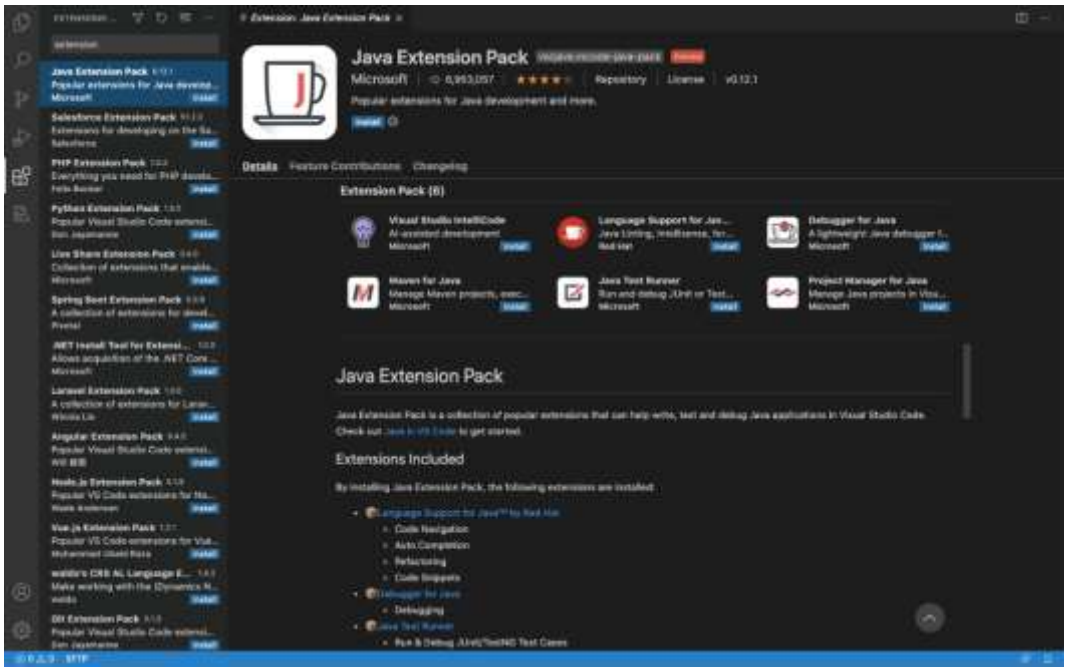
yang hanya digunakan dalam sebuah program, walaupun fungsi itu digunakan berkali-kali. Karena tujuannya untuk penggunaan ulang, umumnya *library* disusun dalam bentuk yang modular, sehingga memudahkan untuk implementasi.

Implementasi dari *library* pun dapat dibedakan menjadi beberapa jenis. Ada *library* yang dipanggil ketika melakukan *build*, maka *library* ini disebut sebagai *static library*. Namun beberapa *library* dipanggil pada *runtime*, dan disebut atau dikategorikan sebagai *dynamic library*. Pada kasus yang terakhir, suatu *dynamic library* mungkin saja tidak dipanggil sama sekali apabila tidak diperlukan dalam suatu eksekusi program, walaupun telah menerima pemrograman khusus untuk itu. Namun, salah satu ciri utama dari *library*, adalah kemampuannya untuk mengimplementasikan *behavior* yang telah disiapkan sebelumnya pada program yang baru, apabila kemampuan tersebut tersedia. Hal ini akan berbeda dengan extensions yang akan dibahas pada subbab berikutnya.

2.2.4 *Extensions*

Ekstensi merupakan komponen program yang menambah kapabilitas baru kepada program yang tidak dimiliki sebelumnya. Ekstensi dapat berupa sebuah paket yang menyediakan fungsi, prosedur, *subroutine*, untuk digunakan dalam sebuah program atau aplikasi. Dalam kaitannya dalam pengembangan perangkat lunak, sebuah ekstensi menambahkan kapabilitas baru pada peralatan pengembangan, atau untuk perangkat lunak yang dikembangkan. Hal ini sedikit beda dengan *library*, yang umumnya menyediakan sesuatu yang telah dapat dilakukan oleh program dasarnya, namun dipaket dalam bentuk yang lebih mudah

untuk digunakan secara berulang-ulang pada program yang berbeda. Gambar 2.3 menunjukkan penggunaan *extensions* pada Visual Studio Code.



Gambar 2.9 Extensions Pada Visual Studio Code

2.2.5 *Object Oriented Programming (OOP)*

OOP atau pemrograman berorientasi obyek, adalah suatu paradigma pemrograman yang mendasarkan konsepnya pada obyek, yang dapat berisi data atau kode. Data dalam bentuk *attribute* atau *properties*, dan kode dalam bentuk prosedur, disebut juga sebagai *methods*. Salah satu fitur utama yang mendasari paradigma ini adalah bahwa obyek memiliki sebuah prosedur sendiri, yang memungkinkannya mengakses dan mengubah data *attribute* yang dimilikinya. Pada OOP, program dirancang untuk berinteraksi satu sama lain dengan basis obyek. Jenis yang paling populer dari paradigma ini adalah yang berbasis kelas,

dimana obyek adalah *instance* yang dibuat dari suatu kelas, yang mana juga menentukan tipenya.

Selain itu, terdapat beberapa fitur-fitur lain dari paradigma OOP yang membedakannya dari pendekatan lainnya, antara lain enkapsulasi, dimana obyek secara khusus menyembunyikan data didalam obyek dari interferensi maupun eksepsi dari luar obyek. Beberapa bahasa pemrograman secara khusus memungkinkan obyek untuk membatasi akses tersebut dengan menggunakan *keyword private*, dan khusus untuk akses dari luar obyek, dilakukan dengan memanfaatkan *methods* yang dideklarasikan dengan *keyword public*.

Fitur lain yang dimiliki oleh OOP adalah *inheritance*. Konsep ini adalah bahwa sebuah obyek *parent* dapat mewariskan *attribute* dan *methods* yang dimilikinya kepada obyek *child*. Obyek turunan ini mungkin saja memiliki *attribute* dan *methods* yang tidak dimiliki obyek *parent*, namun obyek *child* memiliki seluruh *attribute* dan *methods* yang dimiliki oleh *parent*. Selain itu, terdapat pula konsep variabel *static* yang merupakan *attribute* yang dimiliki bersama oleh seluruh obyek, namun hanya ada satu kopi dari *attribute* tersebut. Beberapa bahasa pemrograman mendukung penggunaan OOP secara khusus, namun beberapa bahasa pemrograman membutuhkan beberapa ekstensi untuk mendukung penggunaan paradigma OOP dalam penerapannya.

2.2.6 Komponen Perangkat Keras Pada *Personal Computer* (PC)

Dalam tugas akhir ini, akan dibahas mengenai pengembangan *virtual lab* sebagai media bantu pembelajaran, dengan topik bahasan perbaikan pada

perangkat keras PC. Untuk itu, akan dijabarkan beberapa komponen dari perangkat keras yang dibutuhkan.

a. Display atau Monitor

   Merupakan perangkat untuk menampilkan luaran display. Komponen ini umumnya sering mengalami kerusakan dalam bentuk kerusakan power driver sehingga tidak ada gambar yang tampil di layar, atau tampilan tidak sempurna.

b. Casing

   Melindungi komponen internal yang lebih rawan mengalami kerusakan, dari debu, maupun benturan fisik lainnya.

c. Motherboard

   Merupakan pusat antarmuka dari komponen-komponen lain dari komputer. Biasanya telah memiliki subsistem lain seperti *audio I/O*, *network card*, dan *wireless dongle*.

d. CPU

   Merupakan chip tempat proses terhadap data dilakukan. CPU umumnya didapatkan pada paket yang hanya sesuai dengan model *socket motherboard* tertentu.

e. RAM

   Random Access Memory digunakan sebagai tempat penyimpanan memori *volatile*, yang hanya mampu menyimpan data secara terbatas, namun memiliki kecepatan akses sangat tinggi.

f. Harddisk/Storage

Merupakan tempat penyimpanan data *non-volatile*. Kebalikan dari RAM, kemampuan penyimpanannya relatif jauh lebih besar dibandingkan RAM, namun kecepatan aksesnya relatif jauh lebih rendah dibandingkan akses ke RAM.

g. Power Supply

Merupakan pusat pemasok daya kepada komputer.

h. GPU Card

Merupakan komponen tempat pengolahan data grafis, atau data-data yang optimal untuk dijalankan secara *parallel*.

i. Keyboard dan Mouse

Perangkat antarmuka untuk mendapatkan masukan dari pengguna untuk pengolahan lebih lanjut oleh komputer.

2.2.7 Perangkat Diagnostik

Perangkat diagnostik yang akan digunakan dalam penelitian ini terdiri dari multimeter, *screwdriver*, *network repair* yang terdiri dari *network cable tester*, *user guide* komponen yang menunjukkan spesifikasi masing-masing komponen. Penjelasan masing-masing perangkat adalah sebagai berikut:

a. Multimeter

Sering juga disebut sebagai multitester, adalah sebuah perangkat untuk mengukur tegangan atau arus listrik, dalam bentuk arus AC maupun arus DC. Perangkat ini bekerja dengan sebuah *switch* internal, yang menyambungkan rangkaian instrumentasinya, dengan *probe* yang digunakan terhadap rangkaian pengukuran. Beberapa multitester

melakukan fungsinya secara digital, dan beberapa multitester juga memiliki fungsi tambahan, seperti pengecekan koneksi, pengecekan tahanan, dan pengecekan dioda. Contoh multimeter ditunjukkan pada Gambar 2.8.

Gambar 2.10 Multimeter
(Sumber: http://www.klikglodok.com/perkakas/alat-ukur-multitester/)

b. Screwdriver set

Set obeng ini berfungsi untuk melakukan pembongkaran atau perakitan kembali terhadap komponen yang akan dilakukan pengujian, diagnostik, atau penggantian komponen. Selain itu, perangkat ini dapat

digunakan untuk melakukan perbaikan ringan terhadap kerusakan akibat pemasangan. Komponen ini ditunjukkan pada Gambar 2.9.

Gambar 2.11 Screwdriver Set
(Sumber: https://www.amazon.com/StarTech-com-Precision-Screwdriver-Computer-CTK100P/dp/B0001NYK16)

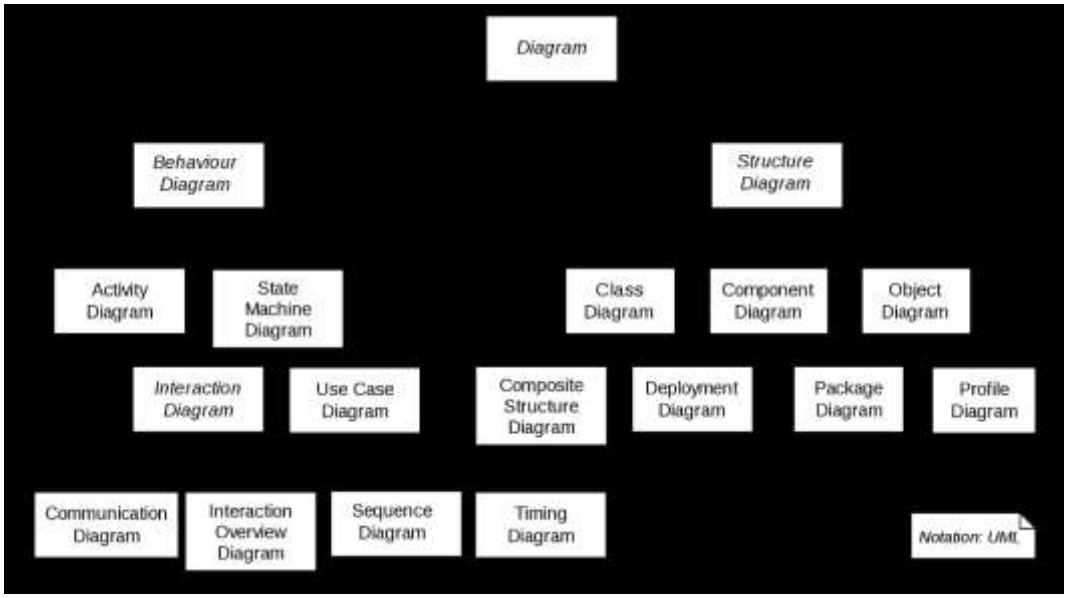
c. Network Cable Tester

Network cable tester, merupakan suatu perangkat yang digunakan untuk melakukan pengujian terhadap konektivitas yang dimiliki oleh *network cable*. Umumnya perangkat ini digunakan untuk mengecek perangakak kabel *network* yang termasuk dalam kategori kapabilitas diagnostiknya. Contoh dari perangkat ini ditunjukkan pada Gambar 2.10.

Gambar 2.12 Network Cable Tester
Source: https://cablematic.com/en/products/network-cable-tester-rj45-rj11-and-rj12-CT006/

2.2.8 Unified Modelling Language

Unified Modelling Language (UML) adalah sebuah bahasa pemodelan serbaguna yang digunakan pada bidang rekayasa perangkat lunak, dengan tujuan untuk memberikan suatu cara yang terstandarisasi untuk memberikan visualisasi atas desain suatu sistem perangkat lunak. Pembuatan UML awalnya memiliki tujuan untuk membuat suatu cara standard dalam melakukan notasi terhadap sistem dan pendekatan dalam perancangan perangkat lunak. Pada tahun 2005, ISO (*International Organization for Standardization*) telah menerbitkan dan menyetujui UML sebagai sebuah standard ISO. Gambar 2.11 menunjukkan jenis-jenis diagram UML.



Gambar 2.13 Diagram UML
(Sumber: https://commons.wikimedia.org/wiki/File:Uml_diagram2.png)

Pada awalnya, UML didesain untuk dokumentasi pendekatan perancangan berorientasi obyek, namun pada perkembangannya, UML telah digunakan pada berbagai macam konteks pemrograman. UML melakukan visualisasi terhadap arsitektur perangkat lunak dalam bentuk diagram, yang memiliki unsur-unsur sebagai berikut:

1. Aktivitas atau pekerjaan
2. Komponen individu dalam sistem dan cara interaksinya dengan komponen perangkat lunak lainnya.
3. Bagaimana sistem akan bekerja
4. Bagaimana suatu entitas akan berinteraksi dengan komponen dan antarmuka lainnya.
5. Antaramuka eksternal.

Pemodelan menggunakan UML memberikan gambaran terhadap suatu model system dengan dua cara:

1. Pandangan statis (atau structural) yang menekankan struktur statis atas sistem, menggunakan obyek, atribut, operasi, dan hubungan. Penerapannya antara lain ada pada UML *class diagram*, dan UML *composite structure diagram*
2. Pandangan dinamis (atau behavioral) yang menekankan sifat dinamis dari sistem yang menunjukkan kolaborasi antara obyek dan perubahan dari *state* internal masing-masing obyek. Contoh penerapannya ada pada *activity diagram* dan *use case diagram*. Pada contoh lebih sempit lagi, ada *sequence diagram* dan *communication diagram*.

2.2.9 Panduan Perbaikan Komputer

Dalam prakteknya, terdapat banyak metode diagnostik dan reparasi terhadap kerusakan komputer. Peralatan, perlengkapan, dan teknik untuk melakukan *troubleshooting* dan perbaikan terhadap komputer. Beberapa panduan telah membahas mengenai teknik-teknik ini dengan mendasarkannya pada teknik yang digunakan oleh para praktisi. Namun, perkembangan teknologi dan teknik manufaktur komputer, memberikan tambahan kompleksitas dalam melakukan diagnosis dan perbaikan terhadap kerusakan komputer. Karena ini, digunakan beberapa buku pedoman dalam referensi perbaikan komputer, yaitu panduan dan buku referensi oleh (McFedries, 2014), dan (Mueller, 2015).