LEMBAR PERSETUJUAN

PERANCANGAN *GAME PUZZLE* UNTUK LATIHAN MEMAHAMI CARA KERJA ALGORITMA

SKRIPSI

Untuk memenuhi sebagian persyaratan untuk mencapai gelar Sarjana Komputer

Disusun oleh :

Anggi Meirania Putri

NIM. 0810680024

Telah diperiksa dan disetujui oleh :

Dosen Pembimbing I	**Dosen Pembimbing II**
Issa Arwani, S.Kom., M.Sc.	**Eriq M. Adams J, ST., M.Kom.**
NIK. 830922 06 1 1 0074	**NIK. 850410 06 1 1 0027**

PERNYATAAN ORISINILITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah SKRIPSI ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata di dalam naskah SKRIPSI ini dapat dibuktikan terdapat unsur-unsur PLAGIASI, saya bersedia SKRIPSI ini digugurkan dan gelar akademik yang telah saya peroleh (SARJANA) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku. (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, Juni 2014

Mahasiswa

Anggi Meirania Putri

NIM. 0810680024

KATA PENGANTAR

Puji syukur kepada Allah SWT, yang telah melimpahkan rahmat dan karunia-Nya sehingga dapat menyelesaikan proposal skripsi dengan judul **Perancangan *Game Puzzle* untuk Latihan Memahami Cara Kerja Algoritma.**

Penulis menyadari bahwa proposal skripsi ini tidak akan dapat terselesaikan dengan baik tanpa keterlibatan dari berbagai pihak, karena itu penulis menyampaikan ucapan terima kasih kepada:

1. Orang Tua, yang telah memberikan dukungan moral dan material.
2. Bapak Issa Arwani, S.Kom., M.Sc. Dan Bapak Eriq M. Adams J, ST., M.Kom. selaku dosen pembimbing.
3. Teman – teman program studi Informatika/Ilmu Komputer Universitas Brawijaya Malang yang telah memberikan masukan kepada penulis dalam menyelesaikan proposal skripsi ini.
4. Dan pihak – pihak lain yang tidak bisa disebutkan satu persatu.

Penulis menyadari bahwa penulisan proposal skripsi ini masih jauh dari kata sempurna. Oleh karena itu penulis mengharapkan masukan berupa saran dan kritik dari semua pihak demi tercapainya kesempurnaan dalam skripsi ini. Akhir kata semoga penulisan skripsi ini dapat memberikan manfaat bagi semua pihak.

Malang, Juli 2014

Penulis

ABSTRAK

Anggi Meirania Putri. 2014. : Perancangan Game Puzzle untuk Latihan Memahami Cara Kerja Algoritma. Skripsi Program Studi Teknik Informatika, Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya, Malang. Dosen Pembimbing : Issa Arwani, S.Kom., M.Sc. dan Eriq M. Adams J., S.T., M.Kom.

Algoritma merupakan kata yang tidak asing lagi untuk para mahasiswa di bidang informatika dan ilmu komputer. Algoritma ini berfungsi sebagai fondasi untuk menyelesaikan suatu masalah secara terstruktur, efektif, dan efisien. Namun dalam penerapannya, banyak mahasiswa di bidang informatika/ilmu komputer yang masih kesulitan untuk memahami cara kerja algoritma. Maka dari itu, dibuatlah *game puzzle* untuk membantu latihan dalam memahami cara kerja algoritma. Perancangan *game* ini diharapkan dapat membantu mahasiswa informatika/ilmu komputer khususnya mahasiswa baru untuk memahami langkah-langkah dalam algoritma. Algoritma direpresentasikan dalam bentuk *flowchart* yang telah dimodifikasi agar dapat menarik minat mahasiswa untuk belajar lebih memahami algoritma.

Perancangan *game* edukasi ini dibuat melalui tahap *game concept design* dan *technical design*. Implementasi perancangan permainan menggunakan bahasa pemrograman C#. Untuk pembuatan objek permainan menggunakan CorelDRAW X4 dan *Unity* 4. Permainan ini dapat diaplikasikan pada *platform Windows* PC. Pengujian aplikasi ini menggunakan metode *White-Box Testing* yang meliputi pengujian unit dan pengujian integrasi. Pengujian pengguna dilakukan dengan membagikan kuisioner kepada responden mahasiswa informatika/ilmu komputer. Dari hasil pengujian didapatkan kesimpulan bahwa permainan ini telah memenuhi kebutuhan fungsional dan non fungsional.

Kata Kunci : *game* edukasi, *puzzle*, algoritma

ABSTRACT

Anggi Meirania Putri. 2014. : ***Puzzle Game Development to Understand The Workings of Algorithm. Undergraduate Thesis of Informatic Study Program, Information Technology and Computer Science Program, Brawijaya University, Malang. Advisor :*** **Issa Arwani, S.Kom., M.Sc.** ***and*** **Eriq M. Adams J., S.T., M.Kom.**

The algorithm is a word that is familiar to the informatics and computer science students. Function of algorithm is as a foundation to resolve a problem in a structured, effective, and efficient. But in practice, informatics / computer science students is still difficulty to understand the workings of algorithm. Therefore, made puzzle game to help understanding the workings of algorithm. The design of this game is expected to help informatics / computer science students, especially new students to understand the steps in algorithm. The algorithm is represented in the form of a flowchart that has been modified in order to attract students to learn and understand the algorithms.

The design of this educational game created by stage game concept design and technical design. Game design implementation using the C# programming language. For the object of making games using CorelDRAW X4 and Unity 4. These games can be applied to the Windows PC platform. This application testing using White-Box Testing that includes the unit testing and integration testing. User testing is done by distributing questionnaires to the respondent informatics / computer science students. From the test results it was concluded that this game has met the functional and non-functional requirements.

Keyword : education game, puzzle, algorithm

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Mahasiswa informatika dan ilmu komputer ditargetkan dapat mengenal dan memiliki pemahaman materi yang salah satunya mengenai algoritma program. Algoritma merupakan kata yang tidak asing lagi untuk para mahasiswa di bidang teknologi informatika dan ilmu komputer. Suatu algoritma adalah urutan instruksi yang tidak ambigu untuk menyelesaikan suatu masalah, yaitu untuk mendapatkan hasil yang disyaratkan untuk setiap input logis dalam suatu jangka waktu tertentu [ZAI-10:3].

Algoritma berguna sebagai fondasi untuk menyelesaikan suatu masalah secara terstruktur, efektif, dan efisien, terutama bagi mahasiswa yang ingin menyusun program komputer untuk menyelesaikan suatu persoalan [SUA-12:1]. Jika komputer mengenal lima data dan kita mengharapkan komputer untuk mengurutkannya, kita harus memberitahu secara detail bagaimana cara mengurutkannya [KAD-05:40-41].

Namun dalam penerapannya, banyak mahasiswa di bidang teknologi informasi terutama mahasiswa baru yang masih kesulitan untuk memahami cara kerja algoritma. Berdasarkan penelitian Dosen Jurusan Pendidikan Teknik Elektro FT UNY Yogyakarta, hanya 69% mahasiswa yang memiliki tingkat pemahaman dan kelulusan materi pemrograman komputer [MUT-08]. Kesulitan yang dihadapi dalam permasalahan ini adalah susahnya mahasiswa mengerti bahasa dari algoritma dan penyelesaian yang harus dikerjakan, serta sulitnya membayangkan struktur data yang akan digunakan [SEM-09].

Pola pikir yang logis dan terurut menjadi dasar cara kerja algoritma. Dibutuhkan cara belajar yang lebih menarik dan menyenangkan, salah satunya dengan permainan. Permainan atau sering disebut dengan *game* merupakan suatu sarana hiburan yang diminati dan dimainkan oleh banyak orang, dari kalangan anak – anak, remaja maupun orang dewasa. Permainan juga dapat berguna untuk melatih dan mengasah kemampuan berpikir seseorang [TAR-12:1]. Otak akan

lebih tertarik pada warna dan bentuk daripada hanya berupa tulisan. Oleh karenanya, guna menimbulkan ketertarikan pada pelajaran maka perlu diberikan pendekatan dari sisi yang sekiranya mereka akan tertarik dengan itu. *Game* digunakan sebagai alternatif media pembelajaran yang dapat memancing minat generasi muda untuk lebih mendalami suatu materi dengan mengangkat tema materi tersebut pada *game* yang dibuat [SAP-11].

Pada penulisan ini penulis ingin membangun suatu *game puzzle*. *Puzzle* memberikan pengalaman edukatif bagi semua umur, bahkan usia lanjut. Selain melatih kesabaran dan konsentrasi, bermain *puzzle* juga dapat meningkatkan kemampuan belajar, khususnya dalam memecahkan masalah [NUG-12]. Dalam skripsi ini, topik yang akan diberikan adalah mengenai algoritma dasar. Dengan *game puzzle*, pengguna dilatih untuk lebih berpikir logis dan dapat mengurutkan *puzzle* secara tepat.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijabarkan, maka dirumuskan beberapa permasalahan:

1. Bagaimana cara memahami algoritma sehingga lebih mudah dimengerti dan lebih menyenangkan?
2. Bagaimana cara mengimplementasikan *game puzzle* yang berisi tentang representasi algoritma?
3. Bagaimana hasil pengujian terhadap perancangan *game puzzle* mengenai representasi algoritma?

1.3 Batasan Masalah

Agar tidak menyimpang dari maksud dan tujuan, maka penyusunan skripsi ini dibatasi menjadi pokok-pokok permasalahan sebagai berikut :

1. *Game* yang dibangun merupakan *game* dua dimensi (2D).
2. Materi algoritma yang dibahas dalam *game* algoritma *puzzle* ini dibatasi pada instruksi utama, yaitu instruksi runtunan, instruksi pemilihan, dan instruksi perulangan.

3. Penyajian algoritma divisualisasikan dalam bentuk gambar yaitu berupa *flowchart* yang telah dimodifikasi.

1.4 Tujuan

Tujuan penyusunan ini adalah membangun suatu *game* yang dapat melatih mahasiswa untuk lebih memahami cara kerja algoritma yaitu melalui *game puzzle* dengan judul "Algoritma *Puzzle*".

1.5 Manfaat

Manfaat dari penulisan skripsi ini sebagai berikut :

a. Bagi Penulis yaitu mendapatkan pengetahuan tentang pembuatan *game puzzle* yang dalam hal ini bertujuan untuk memahami cara kerja algoritma.

b. Bagi Pengguna

1. Mempermudah dan melatih pengguna dalam memahami cara kerja algoritma.
2. Memberikan hiburan yang sekaligus mendidik pengguna permainan menjadi berfikir secara sistematis melalui *game puzzle*.

1.6 Sistematika Penulisan

Sistematika penulisan dalam skripsi ini sebagai berikut :

BAB I Pendahuluan

Memuat latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan.

BAB II Dasar Teori

Memaparkan teori dasar dan teori pendukung yang berhubungan dengan perancangan *game puzzle* serta pemahaman mengenai cara kerja algoritma.

BAB III Metode Penelitian dan Perancangan

Berisi tentang langkah-langkah dalam merancang pembuatan *game*. Serta membahas analisis kebutuhan dan perancangan yang sesuai

dengan teori yang ada dalam membangun *game* agar sesuai dengan keinginan penulis serta bermanfaat bagi pengguna.

BAB IV Implementasi

Membahas tentang implementasi dari *game puzzle* yang dapat dimainkan dengan baik oleh pengguna.

BAB V Pengujian dan Analisis

Memuat hasil pengujian dan analisis sistem *game* yang telah dirancang apakah dapat berjalan dengan baik.

BAB VI Penutup

Memuat kesimpulan yang diperoleh dari hasil akhir implementasi perancangan *game puzzle* dan saran untuk pengembangan lebih lanjut.

BAB II
DASAR TEORI

Bab ini membahas dasar teori yang digunakan untuk menunjang penulisan skripsi mengenai Perancangan *Game Puzzle* untuk Latihan Memahami Cara Kerja Algoritma. Beberapa dasar teori yang dimaksud adalah game yang difokuskan pada *game puzzle*, algoritma dan instruksi utama yang digunakan dalam penyusunan *game*, *flowchart*, *unity*, dan teknik pengujian.

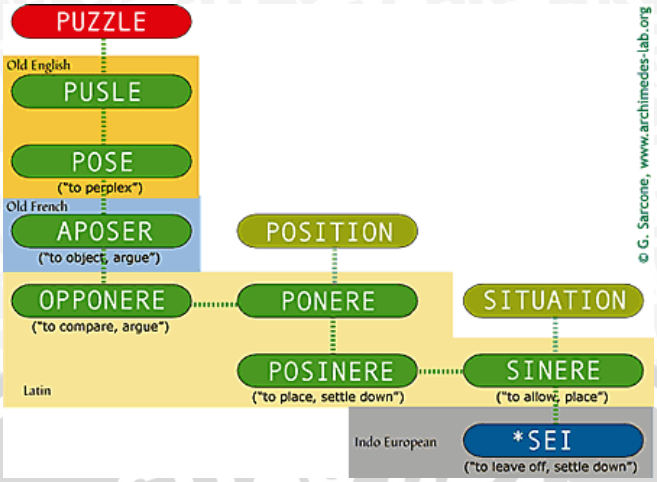
2.1 Permainan Komputer / *Game*

Game atau dalam bahasa Indonesia disebut dengan permainan adalah aktivitas yang dimainkan untuk olahraga atau hiburan menurut aturan. Seperti yang kita ketahui, pada *game* terdapat juga kondisi menang atau kalah. Dalam pembahasan skripsi ini, *game* yang dibahas adalah permainan komputer. Menurut Katie Salen dan Eric Zimmerman, *game* adalah sebuah sistem dimana pemain terlibat dalam konflik buatan, ditentukan oleh aturan tertentu, dan menghasilkan hasil yang terukur [SAL-03:80].

Game dikategorikan menjadi beberapa tipe atau yang disebut dengan *genre* yaitu *action*, *adventure*, *casual*, *educational*, *role-playing game* (RGP), *simulation*, *sports* dan *startegy*. Skripsi ini membahas mengenai *game puzzle* tentang algoritma. *Game puzzle* algoritma adalah permainan yang memiliki *genre educational*.

2.2 *Game Puzzle*

Dari ilmu Etimologi (asal usul kata), *puzzle* awalnya adalah sebuah kata kerja. Kata *puzzle* berasal dari bahasa Perancis Kuno “Aposer”. Kata tersebut dalam bahasa Inggris kuno menjadi “*Pose*” lalu berubah menjadi “*Pulse*” yang merupakan kata kerja dengan arti membingungkan (*bewilder*) atau membaurkan, mengacaukan (*confound*). Sedangkan kata *puzzle* sebagai kata benda merupakan turunan dari kata kerja tersebut. Gambar 2.1 berikut memperlihatkan sejarah kata *puzzle*.



Gambar 2. 1 Asal – Usul kata *puzzle*

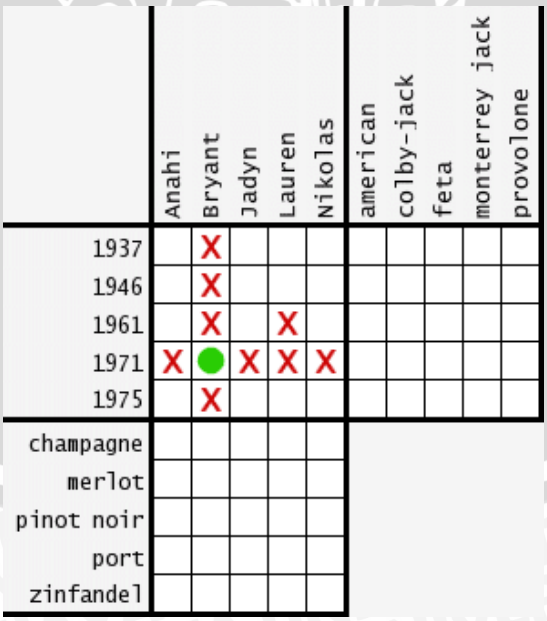
Sumber : www.archimedes.lab.org

2.2.1 Jenis *Puzzle*

Ada beberapa jenis *puzzle*, antara lain :

a. Logic Puzzle

Logic puzzle adalah *puzzle* yang menggunakan logika. Gambar 2.2 di bawah ini memperlihatkan contoh *logic puzzle* berupa *grid puzzle*.



Gambar 2. 2 *Logic Grid Puzzle*

Logic puzzle berbeda dengan *puzzle* lainnya yang berfungsi untuk mengisi waktu luang semata. Memiliki fungsi yang sama untuk latihan pikiran. Akan tetapi *logic puzzle* bukan hanya sekedar mengisi waktu luang, banyak praktik termasuk geologi, perkembangan konsep dan ide dari *game* matematika.

Perancangan *game puzzle* dalam skripsi ini menggunakan konsep *logic puzzle*. *Logic puzzle* merupakan *puzzle* dalam bentuk uraian atau gambar yang dapat mengembangkan keterampilan dan melatih untuk memecahkan masalah. *Logic puzzle* dimainkan dengan cara menyusun kepingan *puzzle* hingga membentuk suatu uraian atau gambar yang utuh.

b. Jigsaw Puzzle

Jigsaw puzzle adalah *puzzle* yang merupakan kepingan- kepingan. Disebut dengan *jigsaw puzzle* karena alat untuk memotong menjadi keping disebut dengan *jigsaw*. Sesuai dengan perkembangan jaman dan teknologi, alat untuk memotong *puzzle* pun berubah. Ada yang menggunakan mesin pon, *scrollsaw* bahkan sinar laser.

c. Mechanical Puzzle

Mechanical puzzle adalah *puzzle* yang kepingnya saling berhubungan. Contoh *puzzle* pada *mechanical puzzle* adalah soma *Cube* dan *Chinese wood knots*. Gambar 2.3 di bawah memperlihatkan *puzzle Chinese wood knots*.

Gambar 2. 3 *Chinese wood knots*

d. Construction puzzle

Construction puzzle merupakan kumpulan potongan – potongan yang terpisah, yang dapat digabungkan kembali menjadi beberapa model. *Construction puzzle* yang paling umum adalah blok – blok kayu sederhana berwarna – warni.

e. Combination Puzzle

Combination puzzle adalah *puzzle* yang dapat diselesaikan melalui beberapa kombinasi yang berbeda. *Rubik's Cube* dan *Hanoi Tower* adalah contoh *Combination Puzzle*.

2.2.2 Manfaat *Game Puzzle*

Permainan *puzzle* memiliki banyak manfaat diantaranya :

a. Meningkatkan kemampuan berpikir dan konsentrasi

Saat bermain *puzzle*, akan melatih sel – sel otak untuk mengembangkan kemampuan berpikirnya dan berkonsentrasi untuk menyelesaikan potongan – potongan kepingan gambar tersebut.

b. Melatih koordinasi tangan dan mata

Puzzle dapat melatih koordinasi tangan dan mata untuk mencocokkan keping – keping *puzzle* dan menyusunnya menjadi satu gambar. *Puzzle* juga membantu anak mengenal dan menghafal bentuk.

c. Meningkatkan keterampilan kognitif

Keterampilan kognitif (*cognitive skill*) berkaitan dengan kemampuan untuk belajar dan memecahkan masalah. *Puzzle* adalah permainan yang menarik karena berbentuk gambar dan warna yang menarik pula. Dengan bermain *puzzle* akan melatih pemain untuk mencoba memecahkan masalah yaitu menyusun gambar.

d. Belajar bersosialisasi

Dua orang yang bermain bersama – sama tentunya butuh diskusi untuk merancang kepingan – kepingan gambar dari *puzzle* tersebut. Akan terbentuk rasa untuk saling membantu dan tercipta suasana yang nyaman dan terciptanya interaksi ketika bermain.

2.3 Algoritma

Algoritma merupakan fondasi yang harus dikuasai oleh setiap mahasiswa yang ingin menyelesaikan suatu masalah secara terstruktur, efektif, dan efisien, terutama bagi mahasiswa yang ingin menyusun program komputer untuk menyelesaikan suatu masalah. Masalah dapat berupa apa saja, dengan catatan untuk setiap masalah ada kriteria kondisi awal yang harus dipenuhi sebelum menjalankan algoritma. Algoritma akan dapat selalu berakhir untuk semua kondisi awalyang memenuhi kriteria.

2.3.1 Definisi Algoritma

Berikut ini definisi-definisi algoritma :

a. Teknik penyusunan langkah-langkah penyelesaian masalah dalam bentuk kalimat dengan jumlah kata terbatas tetapi tersusun secara logis dan sistematis.
b. Suatu prosedur yang jelas untuk menyelesaikan suatu persoalan dengan menggunakan langkah – langkah tertentu dsn terbatas jumlahnya.

c. Susunan langkah yang pasti, yang bila diikuti maka akan mentransformasi data input menjadi output yang berupa informasi.

2.3.2 Ciri Algoritma

Donald E. Knuth, penulis beberapa buku algoritma Abad XX, menyatakan bahwa ada beberapa ciri algoritma, yaitu :

a. Algoritma mempunyai awal dan akhir. Suatu algoritma harus berhenti setelah mengerjakan serangkaian tugas. Dengan kata lain, suatu algoritma memiliki langkah yang terbatas.
b. Setiap langkah harus didefinisikan dengan tepat sehingga tidak memiliki arti ganda, dan tidak membingungkan (*not ambiguous*).
c. Memiliki masukan (*input*) atau kondisi awal.
d. Memiliki keluaran (*output*) atau kondisi akhir.
e. Algoritma harus efektif, bila diikuti benar – benar maka akan menyelesaikan persoalan.

2.3.3 Sifat Algoritma

Berdasarkan ciri algoritma yang dipaparkan Donald Knuth dan definisi algoritma, dapat disimpulkan bahwa sifat utama suatu algoritma adalah sebagai berikut :

a. *Input* : Suatu algoritma memiliki *input* atau kondisi awal sebelum dilaksanakan, bisa berupa nilai – nilai peubah yang diambil dari himpunan khusus.
b. *Output* : Suatu algoritma akan menghasilkan *output* setelah dilaksanakan, atau algoritma akan mengubah kondisi awal menjadi kondisi akhir, da mana nilai *output* diperoleh dari nilai *input* yang telah diproses melalui algoritma.
c. *Definiteness* : Langkah – langkah yang dituliskan dalam algoritma terdefinisi dengan jelas sehingga mudah dilaksanakan oleh pengguna algoritma.
d. *Finiteness* : Suatu algoritma harus memberi kondisi akhir atau *output* setelah sejumlah langkah yang terbatas jumlahnya dilakukan terhadap setiap kondisi awal atau input yang diberikan.

e. *Effectiveness* : Setiap langkah dalam algoritma bisa dilaksanakan dalam suatu selang waktu tertentu sehingga pada akhirnya didapatkan solusi sesuai yang diharapkan.
f. *Generality* : Langkah – langkah algoritma berlaku untuk setiap himpunan *input* yang sesuai dengan persoalan yang diberikan, tidak hanya untuk himpunan tertentu.

Suatu algoritma ini bisa disusun dengan menggunakan bahasa sehari – hari. Tapi dalam skripsi ini akan lebih digunakan algoritma yang berkaitan dengan informatika yaitu yang dapat dibaca oleh komputer. Algoritma ini berupa bahasa program.

2.3.4 Struktur Algoritma

Agar algoritma dapat ditulis lebih teratur maka struktur algoritma sebaiknya dibagi ke dalam beberapa bagian. Salah satu struktur yang sering dijadikan patokan adalah sebagai berikut :

a. Bagian Kepala (*Header*) : memuat nama algoritma serta informasi atau keterangan tentang algoritma yang ditulis.
b. Bagian Deklarasi (Definisi Variabel) : memuat definisi nama variabel, nama tetapan, nama prosedur, nama fungsi, tipe data yang akan digunakan dalam algoritma.
c. Bagian Deskripsi (Rincian Langkah) : memuat langkah – langkah penyelesaian masalah, termasuk beberapa perintah seperti baca data, tampilkan, ulangi, yang mengubah data input menjadi output, dan sebagainya.

Berikut ini contoh struktur sebuah algoritma. Algoritma ini akan menerima dua buah angka bulat kemudian menampilkan angka yang lebih besar.

```
Algoritma Lebih_besar
{menerima dua angka kemudian menampilkan angka yang lebih besar}

Devinisi Variabel
     integer angka1, angka2;
Rincian Langkah
     {memasukkan angka}
     write ("Masukkan angka 1 : ");
     read (angka1);
```

```
      write ("Masukkan angka 2 : ");
      read (angka2);
      {periksa yang lebih besar}
      if (angka1 > angka2)
        then write ("yang lebih besar = ",angka1);
        else write ("yang lebih besar = ",angka2);
      endif
```

2.4 Instruksi Utama

Instruksi utama ini merupakan unsur penting dalam algoritma. Instruksi utama ini digunakan untuk mengatur jalannya pelaksanaan algoritma. Instruksi utama terdiri dari instruksi runtunan, instruksi pemilihan, dan instruksi perulangan.

2.4.1 Instruksi Runtunan

Instruksi runtunan (*sequential*) adalah instruksi yang dikerjakan secara beruntun atau berurutan, baris per baris, mulai dari baris pertama hingga baris terakhir, tanpa ada loncatan atau perulangan.

a. Tiap instruksi dikerjakan sekali, satu per satu.
b. Urutan pelaksanaan instruksi sam dengan urutan penulisan algoritma.
c. Instruksi terakhir merupakan akhir dari algoritma.
d. Urutan penulisan instruksi bisa menjadi penting bila diubah dapat menyebabkan hasi yang berbeda.

Berikut contoh algoritma yang menggambarkan instruksi runtunan, dimana urutan penulisan yang berbeda menghasilkan output yang berbeda.

```
Algoritma Runtunan;
{algoritma menampilkan gaji bersih pegawai dengan memasukkan gaji
pokok kemudian menghitung tunjangan sebesar 25%, dan pajak pph
15%}
Definisi Variabel:
      string nama;
      real gajipokok, tunjangan, pajak;
      real gajibersih;

Rincian langkah:
      write ("Masukkan nama pegawai : ");
      read (nama);
      write ("Masukkan gaji pokoknya : ");
      read (gajipokok);

      tunjangan ←0.25 * gajipokok;
      pajak ←0.15 * (gajipokok + tunjangan);
      gajibersih ←gajipokok + tunjangan – pajak;
```

```
write ("Gaji saudara : ",nama);
write ("adalah = ",gajibersih);
```

2.4.2 Instruksi Pemilihan

Instruksi pemilihan adalah instruksi yang dipakai untuk memilih satu aksi dari beberapa kemungkinan aksi berdasarkan suatu persyaratan. Ada dua bentuk instruksi pemilihan yang sering digunakan, yaitu instruksi ***if / then / else*** dan instruksi ***case***.

2.4.2.1 Instruksi if / then / else

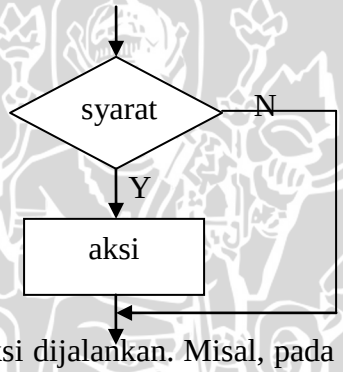
Instruksi *if/ then / else* digunakan untuk memilih alternatif apabila suatu syarat atau kondisi dipenuhi (1 kasus), atau memilih satu alternatif dari dua kemungkinan berdasarkan apakah syarat terpenuhi atau tidak (2 kasus).

Bentuk 1 Kasus

if (syarat)

then aksi

endif



Apabila syarat dipenuhi maka aksi dijalankan. Misal, pada contoh ini nilai x akan ditambah 5 apabila nilainya lebih besar dari 100.

Contoh :

```
if (x > 100)
  then x ← x + 5
endif
```

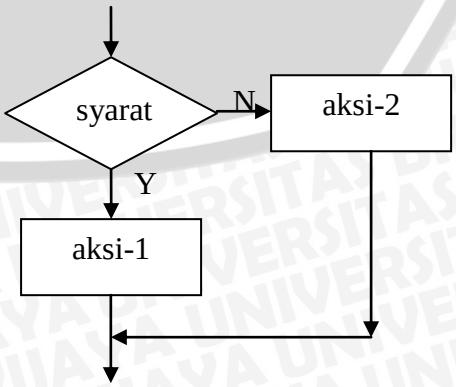
Bentuk 2 Kasus

if (syarat)

then aksi-1

else aksi-2

endif



Apabila syarat dipenuhi maka "aksi-1" dilaksanakan, tetapi bila syarat tidak terpenuhi maka "aksi-2" yang dilaksanakan. Misal, pada contoh berikut ini, bila a lebih besar dari 0 maka akan ditampilkan "bilangan ini positif". Selain itu akan ditampilkan "bilangan ini negatif".

Contoh :

```
if (a > 0)
  then write ("bilangan ini positif")
  else write ("bilangan ini negatif")
endif
```

2.4.2.2 Instruksi *Case*

Instruksi *case* digunakan sebagai instruksi pemilihan dimana aksi yang akan dilakukan bergantung pada nilai dari satu macam variabel saja. Dengan kata lain, variabel yang menetukan pilihan aksi mungkin memiliki banyak macam nilai dan setiap nilainya berkaitan dengan satu macam aksi.

Bentuk Instruksi *Case* :

case (variabel)

nilai-1 : aksi-1;

nilai-2 : aksi-2;

nilai-3 : aksi-3;

.....

default : aksi-n;

endcase

Bentuk *case* diatas dapat diterjemahkan sebagai berikut :

a. Dimungkinkan ada n-buah aksi, mulai dari aksi-1 hingga aksi-n.
b. Setiap aksi hanya dilakukan bila suatu nilai variabel dicapai sesuai dengan persyaratan. Misalnya, aksi-1 dilakukan hanya bila variabel bernilai nilai-1, aksi-2 dilaksanakan bila variabel bernilai nilai-2, dan seterusnya.
c. Apabila tidak ada satupun nilai variabel yang cocok maka aksi-n dikerjakan sebagai aksi "default" (hanya dikerjakan apabila tidak ada yang memenuhi syarat).

Contoh :

```
Algoritma Gaji_Karyawan
```

```
{algoritma yang menerima nama, golongan serta jam kerja kemudian
menampilkantotal gaji yang diterima karyawan}

Definisi Variabel
      real gaji, total, jamkerja, lembur, upah;
      string nama;
      char golongan;

Rincian Langkah
      write ("masukkan nama karyawan : ");
      read (nama);
      write ("masukkan golongannya : ");
      read (golongn);
      write (masukkan jam kerjanya : ");
      read (jamkerja);

      case (golongan)
            ' A ' : upah ← 5000;
            ' B ' : upah ← 6000;
            ' C ' : upah ← 7500;
            ' D ' : upah ← 9000;
           default : write ("golongannya salah!");
                     Upah ← 0;
      endcase

      if (jamkerja > 150)
          then lembur ← (jamkerja – 150 ) * upah * 1.25;
            gaji ← 150 * upah;
          else lembur ← 0;
            gaji ← jamkerja * upah;
      endif
      total ← gaji + lembur;
      write ("Gaji yang diterima saudara : "nama, "adalah = Rp.
",total);
```

2.4.3 Instruksi Perulangan

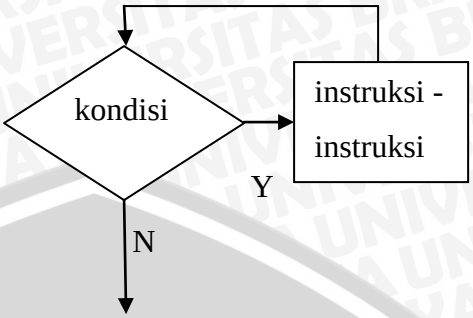
Instruksi perulangan (*repetition*) adalah instruksi yang dapat mengulangi pelaksanaan sederetan instruksi lain berulang kali sesuai dengan persyaratan yang ditetapkan. Struktur instruksi perulangan pada dasarnya terdiri dari :

a. Kondisi perulangan : suatu kondisi yang harus dipenuhi agar perulangan dapat terjadi.
b. Badan (*body*) perulangan : deretan instruksi yang akan diulang – ulang pelaksanaannya.
c. Pencacah (*counter*) perulangan : suatu variabel yang nilainya harus berubah agar perulangan dapat terjadi dan pada akhirnya membatasi jumlah perulangan yang dapat dilaksanakan.

2.4.3.1 Perulangan *while – do*

Bentuk umum :

while (kondisi) do
.....
Instruksi – instruksi
.....
endwhile



Makna dari bentuk instruksi tersebut adalah ulangi ... instruksi – instruksi ... selama kondisi yang diberikan masih terpenuhi.

Perhatikan :

a. Ada instruksi yang berkaitan dengan kondisi sebelum masuk ke *while / do* sehingga kondisi ini benar (terpenuhi) dan pengulangan bisa dilaksanakan. Bila tidak, kemungkinan istruksi *while* tidak bisa dijalankan.
b. Ada satu instruksi di antara instruksi – instruksi yang diulang yang mengubah nilai variabel perulangan agar pada satu saat kondisi perulangan tidak terpenuhi sehingga perulangan bisa berhenti, atau jumlah perulangan bisa dibatasi. Bila tidak, ada kemungkinan perulangan berlangsung terus tak berhingga.

Contoh :

```
Algoritma Perulangan
{mencetak angka 1 hingga 100}

Deklarasi
      integer angka;
Deskripsi
      angka ← 1;
      while (angka < 101) do
            write (angka);
            angka ← angka + 1;
            Endwhile
```

Algoritma tersebut menggunakan *while / do* untuk menampilkan angka 1 hingga 100 secara berurutan. Perhatikan bahwa perintah : angka ← 1 adalah kondisi awal, sedangkan instruksi : angka ← angka + 1 adalah instruksi yang bisa mengubah kondisi hingga tidak terpenuhi, pada saat angka > 100.

2.4.3.2 Perulangan *repeat – until*

Bentuk umum :

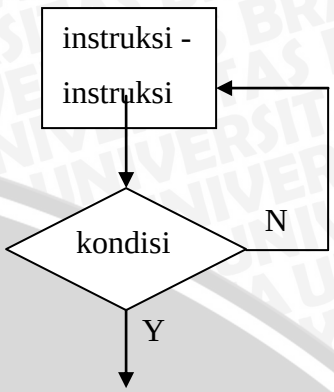
repeat

.....

Instruksi – instruksi

.....

until (kondisi)



Makna dari bentuk instruksi tersebut adalah ulangi pelaksanaan instruksi – instruksi hingga kondisi terpenuhi.

Perhatikan :

a. Instruksi – instruksi akan diulang hanya apabila kondisi tidak terpenuhi, dan ketika kondisi terpenuhi maka perulangan berhenti.
b. Instruksi – instruksi dikerjakan terlebih dahulu sebelum kondisi diperiksa.
c. Harus ada satu instruksi yang mendahului *repeat / until* agar kondisi tidak terpenuhi sehingga perulangan bisa berlangsung.
d. Harus ada instruksi di antara instruksi yang diulang sehingga pada akhirnya dapat mengubah kondisi menjadi terpenuhi dan perulangan berhenti.
e. Apabila di awal pelaksanaan kondisi sudah terpenuhi maka instruksi – instruksi paling tidak dikerjakan satu kali.

Contoh :

```
Algoritma Perulangan
{memakai repeat - until untuk menampilkan Halo sebanyak 25 kali}
Definisi Variabel
     integer cacah;
Rincian Langkah
     cacah ← 1;
     repeat
          write ("Halo...");
          cacah ← cacah + 1;
     until (cacah > 25)
```

2.4.3.3 Perulangan *for*

Bentuk umum :

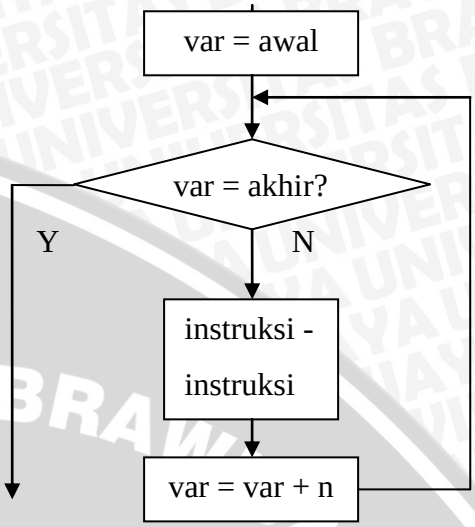
ror (var = *awal* **to** *akhir* **step** n)

.....

Instruksi – instruksi

.....

endfor



Maknanya, ulangi instruksi – instruksi tersebut berdasarkan variabel perulangan mulai dari nilai awal hingga nilai akhir dengan perubahan nilai sebesar n.

Perhatikan :

a. Variabel perulangan (var) harus bertipe dasar (*integer*, *real*, atau *char*).
b. Nilai awal harus lebih kecil dari akhir bila n > 0 (positif).
c. Nilai awal harus lebih besar dari akhir bila n < 0 (negatif).
d. Mula – mula variabel var bernilai awal, kemudian setiap satu kali putaran maka nilai var bertambah sebesar n.
e. Perulangan akan berhanti apabila nilai var sudah mencapai akhir.

Contoh :

```
Algoritma Perulangan
{menampilkan Halo... sebanyak 10 kali dengan memakai instruksi
for}
Deklarasi
      integer cacah;
Deskripsi
      for (cacah = 1 to 10 step 1)
            write ("Halo...");
endfor
```

2.5 Flowchart

Algoritma dapat disajikan dengan menggunakan dua teknik, yaitu teknik tulisan dan gambar. Penyajian algoritma dalam bentuk tulisan direpresentasikan menggunakan metode *English structure* atau *pseudocode*, sedangkan penyajian

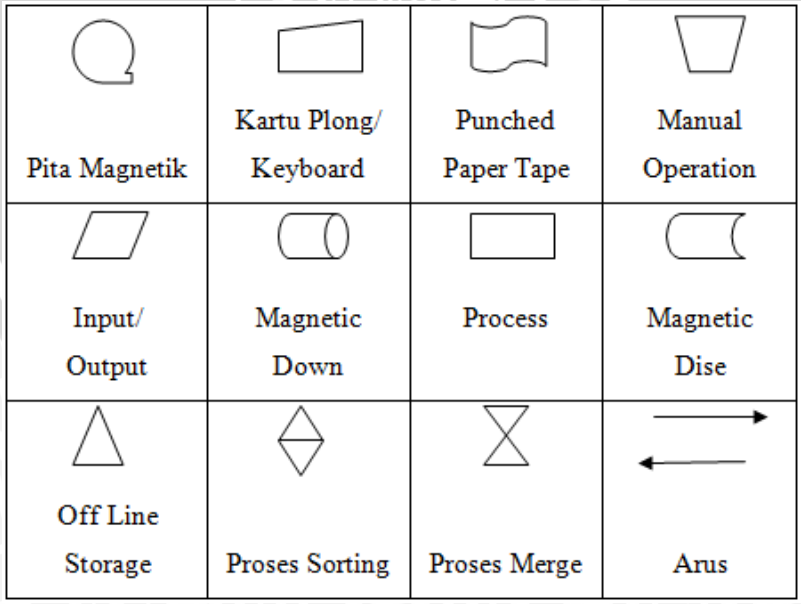
algoritma dengan teknik gambar direpresentasikan menggunakan metode *structure cart*, *Hierarchy plus Input Process – Output* (HIPO), *flowchart*, dan *Nassi Schneiderman chart*. Pada penulisan ini akan dibahas mengenai *flowchart* yang akan digunakan dalam implementasi pembuatan *game puzzle*.

Flowchart digunakan untuk menggambarkan suatu tahapan penyelesaian masalah secara sederhana, terurai, rapi, dan jelas dengan menggunakan simbol – simbol standar. Tahap penyelesaian masalah yang disajikan harus jelas, sederhana, efektif, dan tepat. Dalam penulisan *flowchart* dikenal dua model, yaitu *sistem flowchart* dan *program flowchart*.

2.5.1 Sistem Flowchart

Sistem *flowchart* merupakan diagram alir yang menggambarkan suatu sistem peralatan komputer yang digunakan dalam proses pengolahan data serta hubungan antar peralatan tersebut.

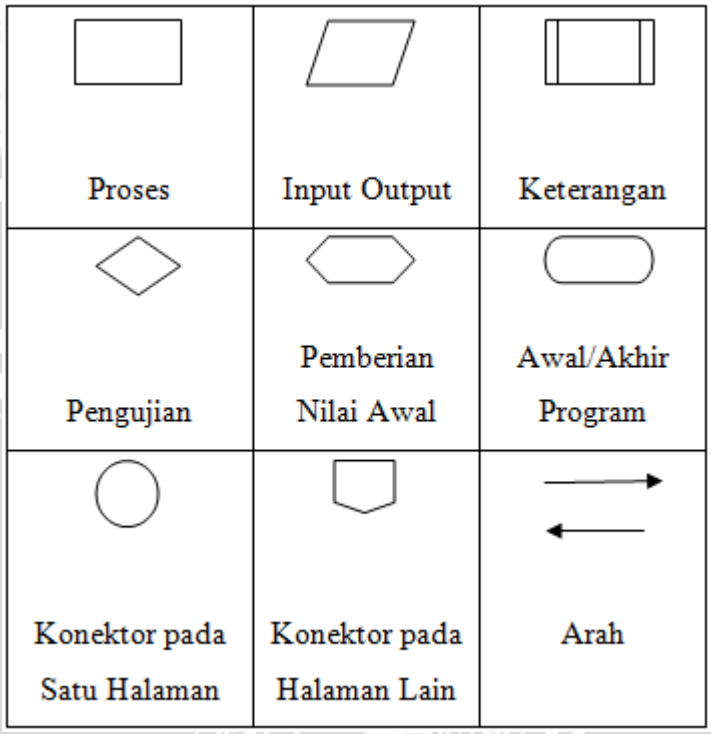
Sistem *flowchart* ini tidak digunakan untuk menggambarkan urutan langkah untuk memecahkan masalah, tetapi hanya untuk menggambarkan prosedur dalam sistem yang dibentuk. Berikut Gambar 2.4, simbol – simbol sistem *flowchat*.



Gambar 2. 4 Simbol – simbol sistem *flowchart*

2.5.2 Program Flowchart

Program *flowchart* merupakan diagram alir yang menggambarkan urutan logika dari suatu prosedur pemecahan masalah. Berikut Gambar 2.5, simbol – simbol standar yang digunakan pada program *flowchart*.



Gambar 2. 5 Simbol – simbol program *flowchart*

Pada penggambaran program *flowchart* terdapat dua jenis metode, yaitu *conceptual flowchart* dan *detail flowchart*. *Conceptual flowchart* menggambarkan tentang alur dari suatu pemecahan masalah secara global saja, sedangkan detail *flowchart* menggambarkan alur pemecahan masalah secara detail. Detail *flowchart* merupakan representasi yang umum digunakan untuk penyajian algoritma.

2.6 *Unity*

Unity merupakan salah satu *game engine* yang banyak digunakan. *Unity* menyediakan fitur pengembangan *game* dalam berbagai platform, yaitu *Unity Web, Windows, Mac, Android, iOS, Xbox, Playstation* 3 dan*Wii*.

Versi gratis *unity* menyediakan fitur pengembangan *game* berbasis *windows, standalone mac*, dan *web*. Sedangkan untuk *platform* lainnya diperlukan lisensi khusus. *Unity pro* juga menyediakan beberapa fitur lebih jika dibandingkan

unity free, misalkan adalah efek bayangan pada objek dan efek *water* yang lebih memukau.

Dalam *unity* disediakan berbagai pilihan bahasa pemrograman untuk mengembangkan *game*, antara lain *JavaScript*, C#, dan *BooScript*. Namun meskipun disediakan tiga bahasa pemrograman, kebanyakan *developer* menggunakan *JavaScript* dan C# sebagai bahasa yang digunakan untuk pengembangan *game*. *Unity* mendukung pembuatan *game* 2D dan 3D, namun lebih ditekankan pada 3D. Pengembangan *game* lebih ditekankan pada desain dan tampilan visual daripada pemrograman.

2.7 Pengujian Perangkat Lunak

Pengujian adalah kegiatan yang dilakukan untuk mengevaluasi kualitas produk, untuk meningkatkan dan mengidentifikasi cacat dan masalah. Arsitektur dari perangkat lunak berorientasi objek menghasilkan sekumpulan *layered subsystem* yang mengenkapsulasi kelas-kelas yang berkolaborasi. Setiap elemen sistem (subsistem dan *class*) melakukan fungsi yang membantu untuk mencapai kebutuhan sistem. Hal ini sangat penting untuk menguji sebuah OO *system* pada berbagai macam level yang berbeda dalam sebuah usaha untuk menemukan kesalahan-kesalahan yang mungkin terjadi dari kolaborasi kelas-kelas dan komunikasi subsistem melewati *architectural layer* [PRE-10].

2.7.1 Teknik Pengujian

Pengujian perangkat lunak memerlukan perancangan kasus uji (*test case*) agar dapat menemukan kesalahan dalam waktu singkat dan usaha minimum. Berbagai macam metode perancangan kasus uji telah berevolusi. Metode-metode ini menyediakan pendekatan sistematis untuk pengujian oleh *developer*. Terlebih lagi metode-metode ini menyediakan mekanisme yang dapat membantu memastikan kelengkapan dari pengujian dan menyediakan kemungkinan tertinggi untuk menemukan kesalahan-kesalahan dalam perangkat lunak [PRE-10].

Strategi untuk pengujian perangkat lunak merupakan aktifitas mengintegrasikan metode perancangan kasus uji ke dalam sebuah rangkaian langkah-langkah pengujian yang terencana sehingga menghasilkan perangkat

lunak yang baik. Strategi menyediakan urutan langkah yang harus dilakukan sebagai bagian dari pengujian. Setiap langkah direncanakan kemudian dikerjakan dan ditentukan berapa banyak usaha, waktu dan sumber daya yang dibutuhkan. Strategi untuk melakukan pengujian perangkat lunak, dimulai dari dengan "pengujian kecil" bergerak menuju ke "pengujian besar". Pengujian berorientasi objek akan memulai pengujian dari *unit testing*, bergerak menuju *integration testing* dan berakhir pada *validation testing* [PRE-10]. Teknik atau metode perancangan kasus uji yang digunakan adalah *white-box testing*.

2.7.1.1 White-Box Testing

White-box testing atau *glass-box testing* merupakan sebuah metode perancangan kasus uji yang menggunakan struktur kontrol dari perancangan prosedural untuk memperoleh kasus uji [PRE-10]. Ada dua jenis pengujian yang termasuk *white-box testing* yaitu *basis path testing* dan *control structure testing*.

Pada skripsi ini menggunakan *basis path testing* yang diusulkan pertama kali oleh Tom McCabe [PRE-10]. *Basis path testing* ini memungkinkan perancang kasus uji memperoleh ukuran kompleksitas logis dari sebuah perancangan prosedural dan menggunakan pengukuran ini sebagai pedoman untuk mendefinisikan *basis set* dari jalur eksekusi (*execution path*). *Test care* yang dilakukan untuk menggunakan *basis set* tersebut dijamin untuk menggunakan setiap *statement* di dalam program paling tidak sekali selama pengujian. Sebelum metode *basis path* dapat diperkenalkan, notasi sederhana untuk representasi aliran kontrol yang disebut diagram alir (*flow graph*) harus diperkenalkan. Setiap representasi desain prosedural yang berupa *flow chart* dapat diterjemahkan ke dalam *flow graph*. *Cyclomatic complexity* adalah metriks perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program. Bila metriks ini digunakan dalam konteks metode pengujian *basis path*, maka nilai yang terhitung untuk *cyclomatic complexity* menentukan jumlah jalur independen (*independent path*) dalam *basis test* suatu program dan memberi batas atas bagi jumlah pengujian yang harus dilakukan untuk memastikan bahwa semua *statement* telah dieksekusi sedikitnya satu kali. Jalur independen adalah jalur yang

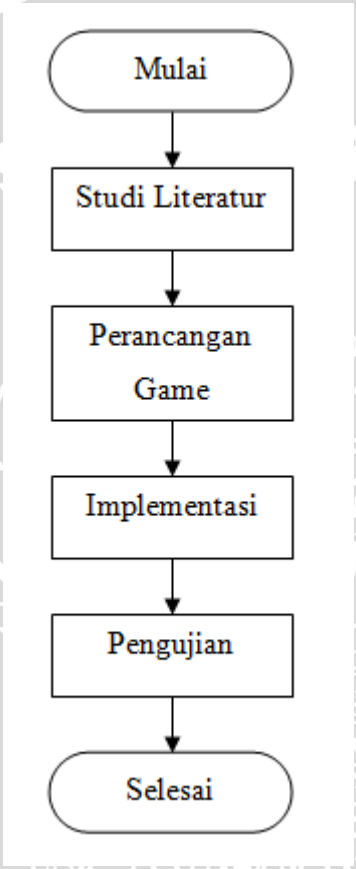
melalui program yang mengenalkan sedikitnya satu rangkaian *statement* proses baru atau suatu kondisi baru. Untuk menentukan *cyclomatic complexity* bisa dilakukan dengan beberapa cara, diantaranya [PRE-10]:

1. Jumlah *region* pada *flow graph* sesuai dengan *cyclomatic complexity*.
2. *Cyclomatic complexity* V(G), untuk grafik G adalah V(G) = E – N + 2, dimana E adalah jumlah *edge*, dan N adalah jumlah *node*.
3. V(G) = P + 1, dimana P adalah jumlah *predicate node* yaitu *node* yang merupakan kondisi (ada 2 atau lebih *edge* akan keluar *node* ini).

BAB III
METODE PENELITIAN DAN PERANCANGAN

3.1 Metode Penelitian

Dalam proposal skripsi ini metode penelitian yang akan digunakan adalah sebagai berikut :



Gambar 3. 1 *Flowchart* Pengerjaan Penelitian

Akan dijelaskan mengenai studi literatur, perancangan *game*, implementasi, dan pengujian pada subbab 3.1.1, 3.1.2, 3.1.3, dan 3.1.4.

3.1.1 Studi Literatur

Studi literatur menjelaskan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori – teori pendukung tersebut meliputi :

1. Definisi permainan (*game*) sebagai sebuah sistem dimana pemain terlibat dalam konflik buatan yang memiliki tujuan , aturan, sistem umpan balik dan pemain.

2. *Game puzzle* menjabarkan tentang jenis – jenis *puzzle* dan manfaat *game puzzle*.
3. Algoritma. Membahas mengenai definisi algoritma, ciri algoritma, sifat algoritma dan struktur algoritma.
4. Instruksi utama merupakan unsur penting algoritma yang digunakan untuk mengatur jalannya pelaksanaan algoritma. Instruksi utama meliputi instruksi runtunan, instruksi pemilihan dan instruksi pengulangan.
5. *Flowchart*. *Flowchart* yang dibahas meliputi sistem *flowchart* dan program *flowchart*.
6. *Asset Maker* yang digunakan adalah *Unity* 4 dan *CorelDRAW* X4.

Penggunaan teknik *white-box testing* untuk pengujian perangkat lunak.

3.1.2 Perancangan Game

Perancangan dalam sebuah permainan merupakan proses merancang atau mendesain sebuah permainan, yang isinya adalah kebutuhan serta prosedur teknis permainan yang akan dibuat. Tujuan dari desain permainan adalah untuk memberikan gambaran secara umum kepada pengguna tentang permainan yang akan dibuat. Perancangan permainan dibagi menjadi 2 jenis yaitu *game concept design* dan *technical design*.

Untuk merancang sebuah desain konsep permainan langkah – langkah yang dilakukan meliputi [GRI-09]:

1. *Data Structure and Lists*, untuk mencari literatur serta mencari informasi yang akan digunakan dalam pembuatan *game*. Langkah ini meliputi pembuatan alur cerita beserta ide karakter yang akan dipakai dan menentukan *main object* yang akan dijalankan oleh pemain nantinya. Langkah ini dijelaskan pada *game concept design* dengan membuat *game design document*.
2. *Working with Graphics*, untuk pembuatan grafik objek beserta semua isi yang akan ditampilkan dalam *game puzzle*. Langkah ini meliputi pembuatan grafik dan *background* yang disesuaikan dengan alur cerita yang telah dibuat. Lanhkag ini dijelaskan pada bab implementasi perancangan *game*.

3. *Animation*, untuk membuat animasi objek yang akan digerakkan. Langkah ini memberikan *collider* dan *rigidbody* pada potongan *puzzle* sehingga bisa digerakkan. Langkah ini merupakan bagian dari implementasi perancangan *game*.
4. *Audio*, memberikan suara pada *game puzzle*. Langkah ini merupakan bagian dari implementasi perancangan *game*.
5. *Platformer*, untuk menggabungkan semua komponen yang telah dibuat dan dihubungkan dengan alur cerita yang telah dibuat sebelumnya. Langkah ini digunakan untuk *error checking* sebelum dipublikasikan yang merupakan bagian dari implementasi perancangan *game*.

3.1.3 Implementasi *Game*

Implementasi pembuatan permainan dilakukan dengan mengacu kepada perancangan permainan serta pembuatan aset dan desain. Implementasi yang dilakukan di sini adalah implementasi antarmuka dan implementasi prosedur program. Adapun Tabel 3.1 menunjukkan implementasi dalam pembuatan permainan ini adalah sebagai berikut :

Tabel 3. 1 Implementasi Pembuatan Permainan

Jenis Implementasi	**Nama Program**
Desain *Flowchart* dan Gambar	*CorelDRAW* X4
Bahasa Pemrograman	C#
Pembuatan Materi 2D	*Unity* 4

3.1.4 Evaluasi dan Analisis (Pengujian)

Pengujian sistem merupakan hal terpenting yang bertujuan untuk menemukan kesalahan – kesalahan atau kekurangan – kekurangan pada perangkat lunak yang diuji. Metode pengujian yang akan dilakukan adalah *white-box testing*. Pengujian bermaksud untuk mengetahui perangkat lunak yang dibuat sudah memenuhi kriteria yang sesuai dengan tujuan perancangan perangkat lunak tersebut.

3.2 Perancangan *Game*

Tahap perancangan permainan terdiri dari dua bagian yaitu *game concept design* dan *technical design*.

3.2.1 *Game Concept Design*

Tahap *game concept design* dilakukan dengan pembuatan *game design document*. Pada tahap pembuatan *game design document*, langkah pertama kali yang dilakukan adalah mencari ide. Pencarian ide dapat dilakukan dengan menuliskan ide yang ada atau berdiskusi dengan kelompok orang yang memiliki disiplin ilmu yang berbeda. Kemudian dari ide – ide yang didapatkan akan ditulis apada sebuah catatan yang disebut dengan *brainstorming note* [ROG-10:32].

Pada bab ini dijelaskan pembuatan *game concept design document* yang merupakan detail – detail dalam permainan. Tujuan dibuatnya dokumen ini adalah untuk mengkomunikasikan kepada tim untuk meminimalisir kesalahan pada saat implementasi karena kurang mengerti tentang permainan yang dibuat [ROG-10:75]. *Game concept design document* mencantumkan segala rincian dalam desain pembuatan permainan. Rincian tersebut meliputi konsep utama *game*, kebutuhan teknologi, tujuan permainan, cerita, deskripsi *game play*, kontrol permainan, latar belakang permainan, sudut kamera, layar awal, *game flow*, antarmuka, karakter dalam cerita, level, dan musik. Ini merupakan bagian dari *Data Structure and Lists* pada konsep langkah penyusunan permainan.

3.2.1.1 Konsep Utama *Game*

Konsep uatam *game* ditunjukkan oleh Tabel 3.2 berikut :

Tabel 3. 2 Konsep utama permainan

No.	Elemen	Keterangan
1.	Judul	*Algoritma Puzzle*
2.	*Platform*	*Windows PC*
3.	Target Pengguna	Mahasiswa baru
4.	*Genre* Permainan	*Education, Puzzle*
5.	USP (*Unique Selling*	• Media edukasi memahami algoritma • Gabungan dua *genre* permainan dalam

	Point)	satu aplikasi • *Art* 2D yang *user friendly*

Sumber : Metodologi dan perancangan

3.2.1.2 Kebutuhan Teknologi

Daftar kebutuhan teknologi untuk membangun permainan ditunjukkan pada Tabel 3.2.

Tabel 3. 3 Daftar Kebutuhan Teknologi

No.	Kebutuhan	Kegunaan
1.	Aplikasi *CorelDRAW* X4	Sebagai media untuk menggambar desain 2D
2.	Aplikasi *Unity* 4	Untuk membangun aplikasi
3.	*Windows* PC	Sebagai alat uji perangkat lunak

3.2.1.3 Tujuan *Game*

Tujuan utama dari *game* ini adalah untuk melatih mahasiswa terutama mahasiswa baru dalam memahami cara kerja algoritma. Algoritma dalam *game* ini disajikan dalm bentuk *flowchart* yang telah diacak. Pemain mengurutkan *puzzle* – *puzzle* dari *flowchart* yang diacak tersebut menjadi satu susunan algoritma yang urut.

3.2.1.4 Cerita *Game*

Cerita dimulai ketika pelajar SMA yang sudah lulus dan menginginkan untuk meneruskan kuliah di universitas jurusan informatika. Ia bersemangat untuk dapat masuk ke informatika yang dapat mengasah kemampuannya dalam penggunaan ilmu tentang komputer ini dan ia belajar dengan baik agar dapat masuk informatika.

Namun setelah menjalani awal – awal masa kuliah, dia kaget karena terdapat pelajaran dasar tentang algoritma yang sulit untuk dipahaminya. Akhirnya ia melakukan eksplorasi dengan menemukan permainan yang dapat melatih kemampuannya dalam memahami algoritma, yaitu melalui *game puzzle*.

Akan ada pengantar singkat mengenai pengertian algoritma dan penjelasan singkat mengenai simbol – simbol program *flowchart* yang nantinya akan digunakan dalam setiap level. Level dalam *game* ini dimulai dari algoritma runtunan biasa yang memuat langkah – langkah menghitung beberapa rumus di matematika dan dibatasi sampai tentang instruksi utama yang ada dalam algoritma. Rancangan cerita akan ditunjukkan oleh Gambar 3.2

Gambar 3. 2 Story board

3.2.1.5 Kontrol *Game*

Kontrol *game* menunjukkan bagaimana pemain mengendalikan *game*. Kontrol *game* ini menggunakan *mouse*. *Mouse* merupakan kendali utama dalam *game* ini. *Mouse* yang digerakkan akan menggerakkan *kursor* yang digunakan untuk memilih menu pada menu utama dan melanjutkan *game* tiap levelnya.

3.2.1.6 *Game Camera*

Game ini menggunakan *static camera* atau *locked camera*. *Static camera* tidak mengubah posisi dan tetap diam dalam satu *screen* lokasi. Dengan *static camera* ini, pemain bisa lebih terfokus dengan semua elemen dan objek yang ada dalam satu *screen*, sehingga bisa memaksimalkan unsur *art* dari sebuah *game* tanpa harus bersusah payah menciptakan suatu suasana baru karena hanya dilihat dari sudut pandang yang sama.

3.2.1.7 *Start Screen*

Layar awal merupakan layar pembuka permainan yang berisi logo institusi yang menaungi pembuatan permainan meliputi Logo Universitas Brawijaya yang ditunjukkan pada gambar 3.3. Logo PTIIK yang ditunjukkan pada gambar 3.4 dan Logo *Game Lab* yang ditunjukkan pada gambar 3.5.

Gambar 3. 3 Logo Universitas Brawijaya

Sumber : Metodologi dan Perancangan

Gambar 3. 4 Logo PTIIK

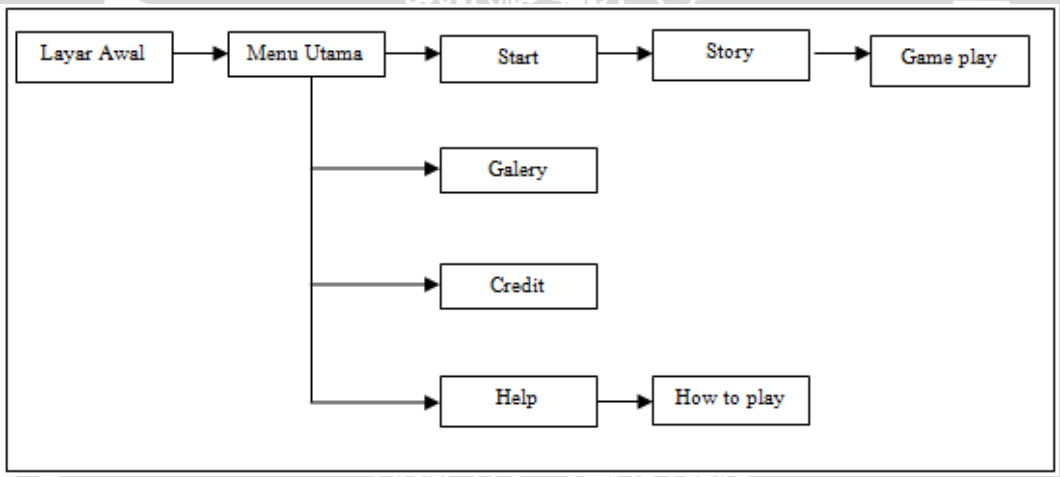
Sumber : Metodologi dan Perancangan

Gambar 3. 5 Logo *Game* Lab

Sumber : Metodologi dan Perancangan

3.2.1.8 *Game Flow*

Game flow merupakan gambaran aliran jalan permainan yang dapat dilihat oleh pemain. Terdapat beberapa layar dalam permainan ini meliputi layar awal, cerita, menu utama, dan hal – hal yang menyangkut permainan ini. *Game flow* ditunjukkan pada Gambar 3.6.



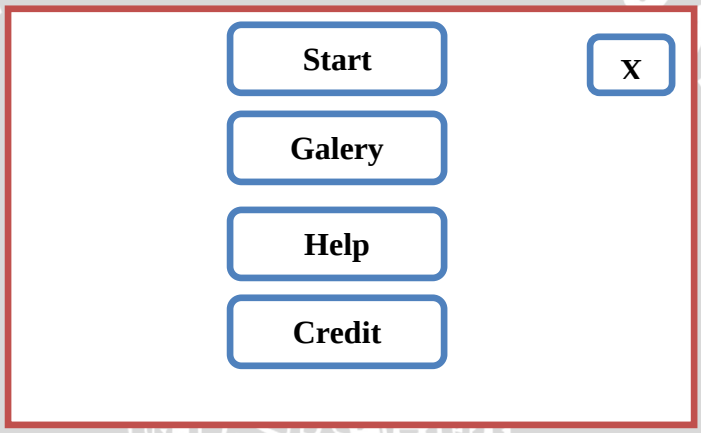
Gambar 3. 6 *Game Flow*

Pada permainan Algoritma *Puzzle* ini, ketika pemain memulai game ini akan ditampilkan *splash* logo Universitas Brawijaya, PTIIK, dan Laboratorium *Game*. Setelah itu pemain memasuki halaman menu utama yang terdiri dari *start*, *galery*, *credit*, dan *help*. Ketika tombol *start* ditekan maka sebelum *game play* terdapat *story* sebagai pengantar cerita. Di *galery*, ada pilihan untuk menyimpan atau tidak hasil Algoritma *Puzzle* dalam bentuk format gambar. Halaman *help* akan menampilkan cara permainan Algoritma *Puzzle*. Dan halaman *credit* akan menampilkan tentang profil pembuatan permainan.

3.2.1.9 *Interface*

Pada permainan algoritma *puzzle* ini dirancang empat *screen* yang terdiri dari menu utama, *help screen*, *credit screen*, *option* permainan *screen*. Rancangan ditunjukkan pada Tabel 3.4.

Tabel 3. 4 Rancangan Antarmuka Menu Utama

Nama Layar	**Main Menu**
Sketsa	
Keterangan	Main menu berisi tombol *play, option, help, credit dan exit*. 1. *start* : Mulai permainan. 2. *galery* : Menyimpan hasil gambar *flowchart*. 3. *help* : Informasi cara bermain. 4. *credit* : berisi informasi permainan serta keterangan tentang pembuat permainan.

3.2.1.10 Karakter Pemain

Dalam permainan Algoritma *Puzzle* ini tidak terdapat karakter pemain ataupun musuh. Namun, sebagai pengantar cerita dari game ini terdapat dua karakter remaja (laki – laki dan perempuan) yang telah lulus dari SMA dan akan masuk ke universitas dengan jurusan informatika sebagai wakil gambaran dari mahasiswa baru informatika yang sebenarnya. Namun diawal pembelajaran, dua

remaja ini merasa kebingungan dengan algoritma. Berikut adalah gambaran kedua mahasiswa baru tersebut. Mahasiswa laki – laki ditunjukkan pada Tabel 3.5 dan mahasiswa perempuan pada Tabel 3.6.

Tabel 3. 5 Mahasiswa Laki – Laki

Sketsa	Profil
	Nama : Aleagro Jenis kelamin : Laki – laki Mimik wajah dalam *game* : Senang, sedih, bingung, berpikir. Keterangan : Remaja laki – laki ini baru saja masuk teknik informatika di sebuah universitas. Ia sangat menyukai berkutat dengan komputer. Terutama bermain *game*.

Tabel 3. 6 Mahasiswa Perempuan

Sketsa	Profil

	Nama : Aleandra Jenis kelamin : Perempuan Mimik wajah dalam *game* : Senang, sedih, bingung, berpikir. Keterangan : Remaja perempuan ini baru saja masuk teknik informatika di sebuah universitas. Ia sangat menyukai berkutat dengan komputer. Terutama hal – hal yang berkaitan dengan desain dan edit gambar.

3.2.1.11 *Level*

Level disini ditunjukkan dengan potongan – potongan *puzzle* yang berbeda – beda. Setelah pengantar tentang pengertian algoritma dan simbol – simbol program *flowchart* untuk merepresentasikan algoritma, pemain dapat melakukan permainan Algoritma *Puzzle* dari level awal sampai akhir. Deskripsi *level* ditunjukkan pada Tabel 3.7.

Tabel 3. 7 Deskripsi *Level*

Level Name	Deskripsi
Level A	Disini akan ditampilkan berupa potongan *puzzle flowchart* algoritma instruksi runtunan sederhana.
Level B	Disini tetap algoritma instruksi runtunan namun dengan proses *flowchart* yang lebih panjang.
Level C	Algoritma disini menggunakan instruksi pemilihan *if/then/else* dengan bentuk 1 kasus.
Level D	Algoritma menggunakan instruksi pemilihan *if/then/else* dengan bentuk 2 kasus.
Level E	Algoritma menggunakan instruksi pemilihan *if/then/else* dengan bentuk bersusun (lebih dari 1 syarat).
Level F	Algoritma menggunakan instruksi pemilihan *case.*
Level G	Algoritma menggunakan instruksi perulangan *while – do.*
Level H	Algoritma menggunakan instruksi perulangan *repeat – until.*
Level I	Algoritma menggunakan instruksi perulangan *for.*

3.2.1.12 Penghargaan

Penghargaan atau yang lebih dikenal '*reward*' yang ada dalam permainan ini diwujudkan dengan kata – kata yang bisa membangkitkan semangat pemain. Seperti "Selamat, kamu berhasil!", dan sebagainya pada saat pemain berhasil memenuhi pola *flowchart* yang telah ditetapkan dan melakukan sesuai dengan kotak dialog yang ditunjukkan. Serta pemain berhasil menyusun *puzzle* dengan benar.

3.2.2 *Technical Design*

Technical design menjelaskan tentang kebutuhan teknis dari game. *Technical design* dari perancangan game ini meliputi pembuatan *use case diagram*, *class diagram* dan *activity diagram*.

3.2.2.1 *Use case* Diagram

Use case diagram dalam perancangan game digunakan untuk memodelkan fungsionalitas dari permainan. Sebelum membuat *use case diagram*, kita harus mengidentifikasi aktor serta kebutuhan terlebih dahulu.

1. Identifikasi Aktor

Tahap ini adalah tahap untuk melakukan identifikasi terhadap aktor yang akan berinteraksi dengan permainan Algoritma *Puzzle*. Tabel 3.8 memperlihatkan aktor yang terlibat beserta penjelasannya yang merupakan hasil dari proses identifikasi aktor.

Tabel 3. 8 Identifikasi Aktor

Aktor	Deskripsi
Pemain	Pemain adalah pengguna yang dapat memainkan permainan, mengontrol permainan, mendapatkan akses untuk melihat info dan bantuan.

2. Daftar Kebutuhan

Daftar kebutuhan adalah daftar yang menjabarkan kebutuhan fungsional yang ada di dalam permainan Algoritma *Puzzle*, ditunjukkan pada Tabel 3.9.

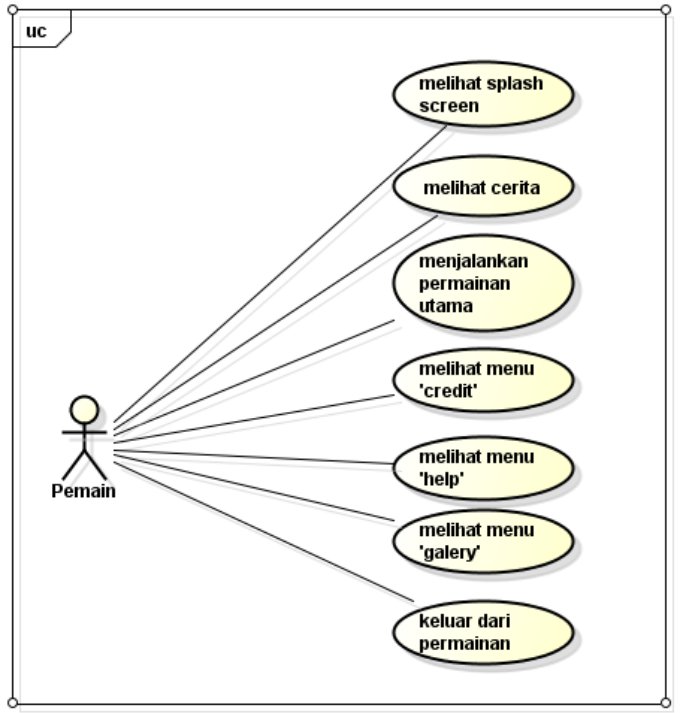
Tabel 3. 9 Deskripsi Daftar Kebutuhan

ID	Kebutuhan	Aktor	Nama *Use Case*
F01	Permainan harus dapat menampilkan splash screen berupa logo UB, PTIIK,	Pemain	Melihat *Splash*

	dan *Game* Lab sebelum masuk ke dalam permainan.		*Screen*
F02	Perangkat lunak harus menyediakan antarmuka untuk menampilkan berbagai menu utama permainan.	Pemain	Melihat menu utama
F03	Permainan harus dapat menampilkan cerita. Dimulai dari pengantar kisah karakter hingga menemukan solusinya melaui *game puzzle*.	Pemain	Melihat cerita
F04	Permainan harus menyediakan fasilitas untuk melihat bantuan, yaitu halaman yang menunjukkan cara kontrol permainan.	Pemain	Melihat bantuan
F05	Permainan harus menyediakan fasilitas untuk melihat *credit*. Sehingga pemain dapat melihat siapa saja yang terlibat dalam pembuatan permainan, referensi pembuatan permainan dan kontak pembuat permainan.	Pemain	Melihat *credit* menu
F06	Permainan harus mampu memulai permainan ketika pemain menekan tombol '*start'*.	Pemain	Menjalankan permainan utama
F07	Permainan harus mampu menyusun permainan *puzzle* dari potongan – potongan *flowchart* yang sudah diacak agar menjadi satu algoritma utuh.	Pemain	Menjalankan serta menyusun *puzzle*

3. *Use Case Diagram*

Use case diagram dalam perancangan game digunakan untuk memodelkan fungsionalitas dari permainan. Diagram *use case* ini melibatkan *user* sebagai aktor dan beberapa *use case* yang dikerjakan aktor.



Gambar 3. 7 *Use Case Diagram Game* Algoritma *Puzzle*

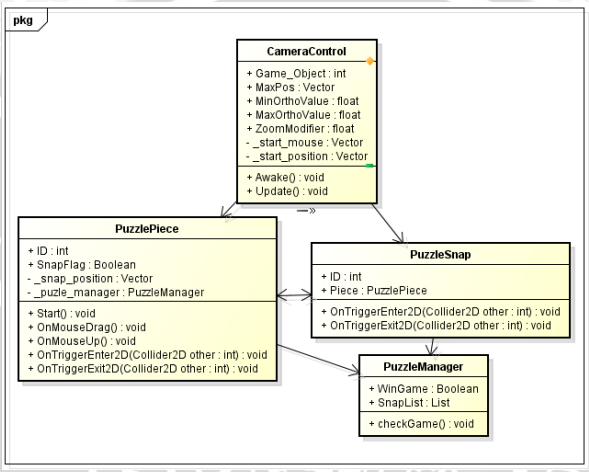
Dari *use case* di atas dapat diketahui bahwa pemain dapat melakukan beberapa hal antara lain melihat *splash screen*, melihat cerita, menjalankan permainan utama, melihat menu *credit*, *help*, dan *galery*, dan keluar dari permainan.

Splash screen merupakan tampilan dari logo Universitas Brawijaya, Logo Program Teknologi Informasi dan Ilmu Komputer (PTIIK), dan logo Laboratorium Game Universitas Brawijaya. Sebelum menjalankan permainan utama, akan ditampilkan pengantar cerita tentang dua mahasiswa yang membicarakan tentang masuk informatika dan kesulitan mempelajari algoritma serta solusi melalui permainan Algoritma *Puzzle*.

Pada menu *galery* akan ditampilkan hasil gambar *flowchart* dari *puzzle* yang sudah dimainkan dengan benar, sehingga mahasiswa dapat belajar lagi dari gambar tersebut. Pada menu *help*, akan ditampilkan petunjuk-petunjuk singkat untuk melakukan permainan. Pada menu *credit*, akan ditampilkan sekilas tentang pembuat permainan Algoritma *Puzzle*. Bila telah selesai, pemain dapat keluar dari permainan.

3.2.2.2 *Class Diagram*

Class diagram memberikan gambaran pemodelan elemen – elemen *class* yang membentuk sebuah perangkat lunak. *Class* didapatkan dengan menganalisis secara detail terhadap *use case* yang dimodelkan. Deskripsi dari *class diagram* permainan ini ditunjukkan pada gambar 3.8 dan tabel 3.10.



Gambar 3. 8 *Class Diagram*

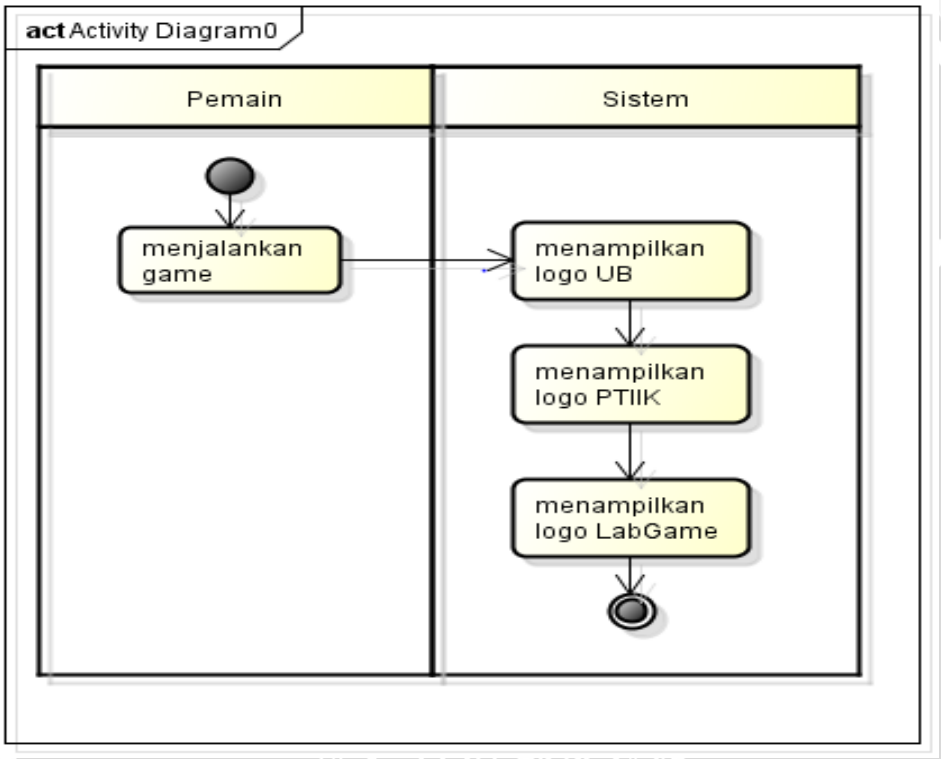
Tabel 3. 10 Deskripsi *Class Diagram*

Class	Keterangan
`PuzzlePiece`	*Class* untuk mengatur objek *puzzle piece* dalam hal melakukan tumbukan
`PuzzleSnap`	*Class* untuk mengatur *puzzle snap* dalam hal tumbukan
`PuzzleManager`	*Class* untuk mengatur agar *PuzzlePiece* dan *PuzzleSnap* sehingga dapat saling menyatu bila *puzzle* cocok
`CameraControl`	*Class* untuk mengatur tampilan kamera

3.2.2.3 *Activity Diagram*

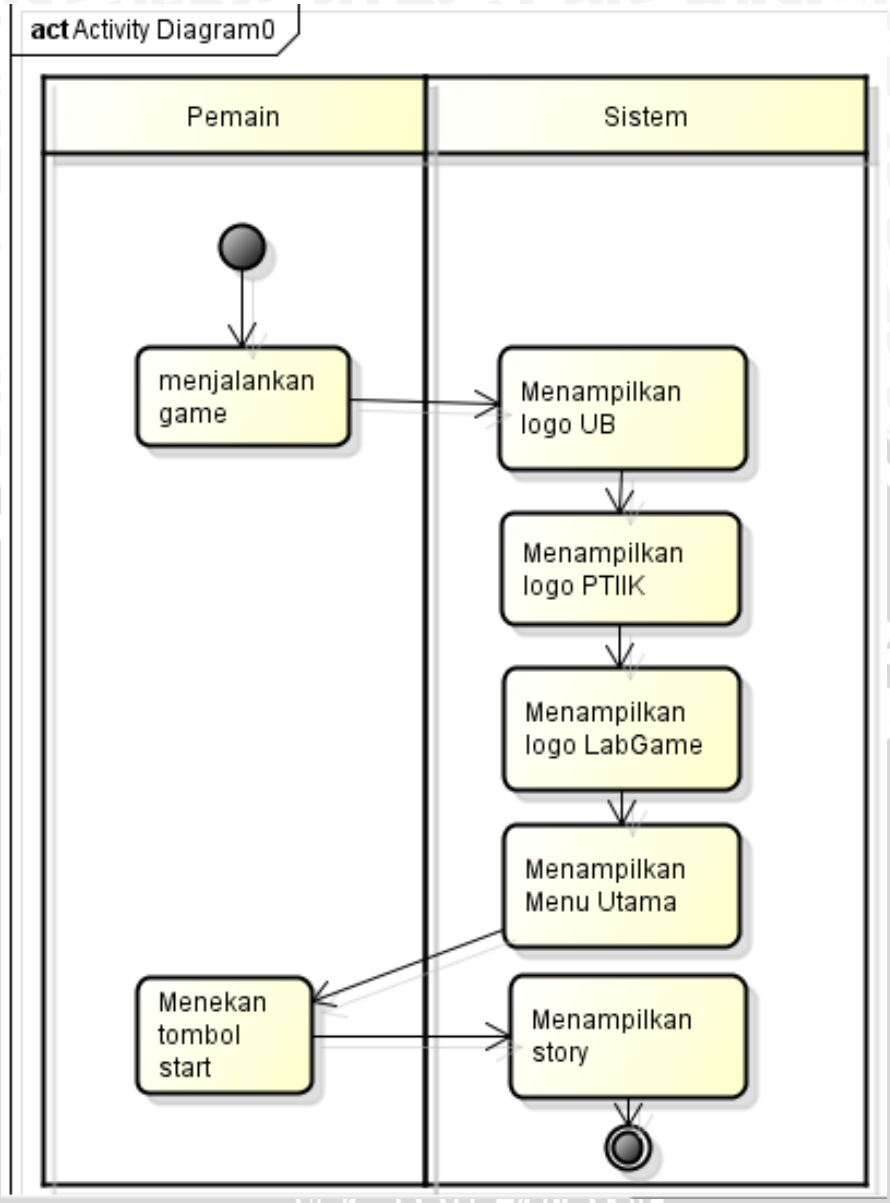
Pembuatan *activity diagram* ini bertujuan untuk menggambarkan urutan aktivitas dari proses pada setiap *use case* yang ada. Berikut merupakan gambar

dari masing – masing *activity diagram*. Gambar 3.9 menunjukkan *activity diagram* ketika menjalankan *splash screen*, gambar 3.10 menunjukkan *activity diagram* melihat *story*, gambar 3.11 menunjukkan *activity diagram* ketika melihat *help screen*, gambar 3.12 menunjukkan *activity diagram* ketika melihat *option screen* (dimana *option screen* berfungsi untuk mengatur *volume sound*), gambar 3.13 menunjukkan *activity diagram* ketika menjalankan permainan utama dan gambar 3.14 menunjukkan *activity diagram* melihat menu *credit*.



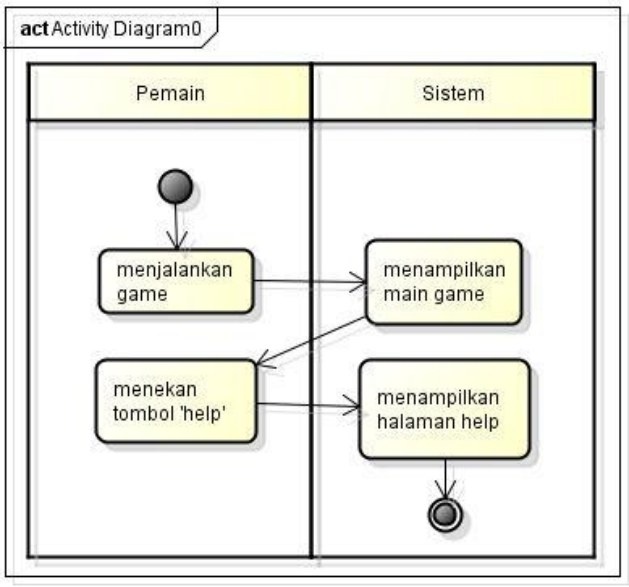
Gambar 3. 9 *Activity Diagram Splash Screen*

Untuk melihat tampilan *splash screen*, aktivitas yang dilakukan yaitu menjalankan *game* yang dilakukan oleh pemain, maka sistem akan langsung membaca perintah tersebut dan menampilkan logo secara urut dari logo Universitas Brawijaya, logo PTIIK, dan logo laboratorium *game*.



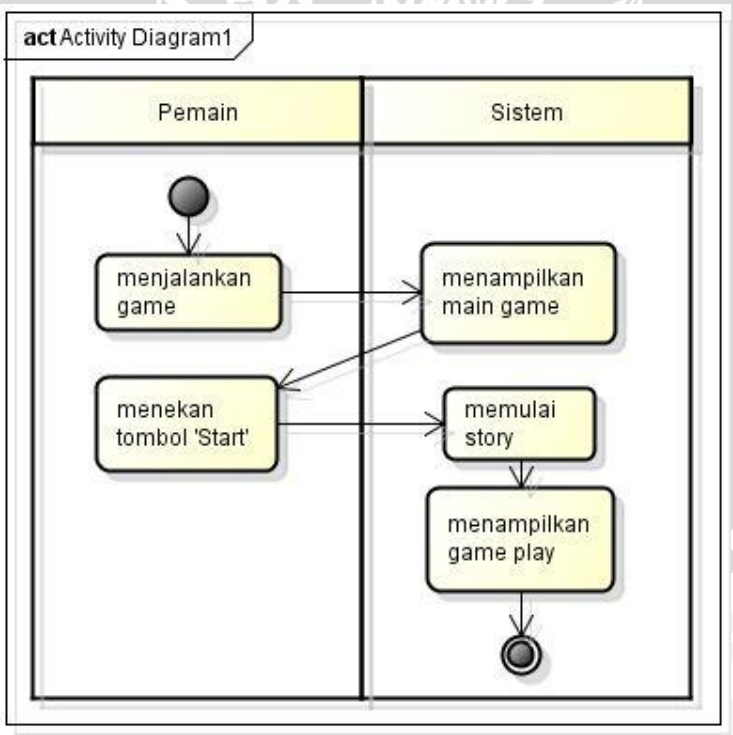
Gambar 3. 10 *Activity Diagram* melihat *story*

Pada activity diagram melihat *story*, setelah pemain menjalankan dan menampilkan maka akan diproses oleh sistem sehingga masuk ke menu utama. Pemain menekan tombol *start* untuk memulai permainan. Sebelum ke permainan utama, sistem akan menampilkan *story* antara Aleagro dan Aleandra.



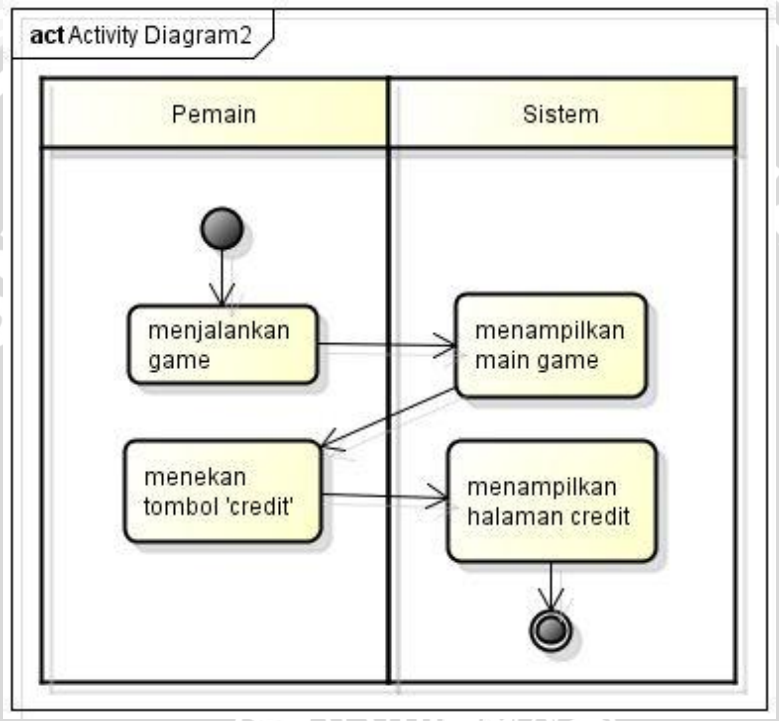
Gambar 3. 11 *Activity Diagram* melihat *help screen*

Pada menu utama yang ditampilkan sistem, ketika pemain menekan tombol *help*, maka sistem akan membaca dan menampilkan halaman *help*. Halaman ini berisi tentang petunjuk singkat dalam permainan Algoritma *Puzzle*.



Gambar 3. 12 *Activity Diagram* memulai permainan

Activity diagram memulai permainan merupakan bagian utama dari semua perancangan permainan *puzzle* ini. Pertama sistem akan menampilkan inisialisasi permainan, setelah itu sistem akan menuju permainan yaitu bagian menu utama. Pemain harus menekan tombol *start* dulu agar masuk dalam permainan. Namun sebelum bermain *puzzle*, maka sistem akan menampilkan cerita pengantar. Setelah itu pemain dapat menyelesaikan permainan.



Gambar 3. 13 *Activity Diagram* melihat *credit screen*

Ketika pemain ingin melihat halaman *credit*, maka pemain harus membuka permainan *puzzle*. Maka sistem akan menampilkan menu utama. Pemain menekan tombol *credit* untuk menampilkan halaman *credit*. Halaman *credit* berisi tentang identitas singkat pembuat *game* dan kontak pembuat yang bisa dihubungi.

BAB IV
IMPLEMENTASI

Bab ini akan membahas implementasi dari pembuatan *game puzzle* untuk mempermudah memahami algoritma yang direpresentasikan dengan *flowchart* sesuai dokumen desain permainan.

4.1 Spesifikasi Sistem

Perangkat lunak ini dikembangkan dalam lingkungan implementasi yang terdiri dari perangkat keras dan perangkat lunak.

4.1.1 Spesifikasi Lingkungan Perangkat Keras

Spesifikasi perangkat keras yang dipakai dalam proses pengembangan dijelaskan pada Tabel 4.1.

Tabel 4. 1 Spesifikasi lingkungan perangkat keras komputer

	PERSONAL COMPUTER (PC)
Processor	*Intel(R) Pentium(R) Dual-Core CPU T4200 (2.0 GHz, 800 MHz FSB, 1 MB L2 cache)*
Memory (RAM)	1 GB
Harddisk	250 GB HDD
Motherboard	*Acer Aspire* 4736Z
Graphic Card	*Intel*

4.1.2 Spesifikasi Lingkungan Perangkat Lunak

Spesifikasi perangkat lunak yang dipakai dalam proses pengembangan perangkat lunak dijelaskan pada Tabel 4.2.

Tabel 4. 2 Spesifikasi lingkungan perangkat lunak komputer

	SOFTWARE
Operating System	*Microsoft Windows 7 Ultimate 32-bit (6.1, build 7600)*
DirectX Version	*DirectX 11*
Programming Language	C#

Graphics Editor	*CorelDRAW X4*
Integrated Development Environment	*Unity 4*

4.1.3 Batasan – Batasan dalam Implementasi

Beberapa batasan dalam mengimplementasikan perangkat lunak ini adalah sebagai berikut :

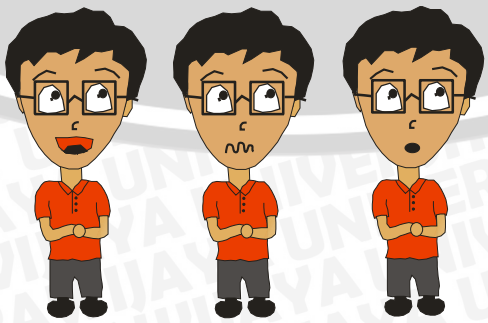
1. Aplikasi *game puzzle* ini dirancang untuk *windows PC*.
2. Permainan merupakan *single player*.
3. Fitur yang dibahas pada implementasi aplikasi ini adalah :
   a. *Puzzle* algoritma dasar hanya sampai perintah instruksi utama yang direpresentasikan dalam bentuk *flowchart*.
   b. Konsep jalan cerita yang disesuaikan dengan permainan.
4. Permainan menggunakan bahasa Indonesia.
5. Permainan yang dibangun merupakan permainan 2D.

4.2 Implementasi Antarmuka

Implementasi antarmuka terdiri dari implementasi karakter, *main menu*, *gallery screen*, dan permainan utama.

4.2.1 Implementasi Karakter Aleagro

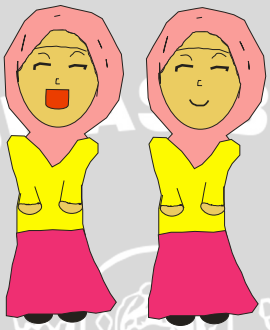
Karakter utama bernama Aleagro, didesain bertubuh sedang, berkacamata, dengan mengenakan baju berwarna merah dan celana abu-abu gelap sebagai representasi dari mahasiswa informatika. Desain mimik wajah dikategorikan menjadi 3 bagian besar yaitu ekspresi senang, sedih dan bingung. Tampilan karakter Aleagro ditunkukkan pada Gambar 4.1.



Gambar 4. 1 Aleagro

4.2.2 Implementasi Karakter Aleandra

Aleandra merupakan karakter pendukung dari Aleagro yang memberi semangat. Mengenakan pakaian berwarna kuning, dengan bawahan pink dan berkerudung pink. Aleandra didesain dengan dua mimik wajah yaitu senang tertawa dan senyum penyayang. Tampilan karakter Aleandra ditunjukkan pada Gambar 4.2.



Gambar 4. 2 Aleandra

4.2.3 Implementasi *Main Menu*

Main menu merupakan halaman menu yang akan tampil pertama kali ketika permainan dijalankan. Di halaman *main menu*, *button* yang bisa dipilih untuk menuju menu selanjutnya yaitu :

- ***Start***, *button* ini digunakan untuk masuk ke main permainan.
- ***Galery***, *button* ini digunakan untuk melihat hasil *flowchart* yang sudah berhasil disimpan setiap *level*nya.
- ***Help***, *button* ini digunakan untuk masuk ke *help screen*.
- ***Credit***, *button* ini digunakan untuk masuk ke *credit screen*.
- ***Exit***, *button* ini digunakan untuk keluar dari permainan.

4.2.4 Implementasi *gallery screen*

Tampilan halaman *gallery* memberikan data gambar *flowchart* yang sudah benar prosesnya pada setiap *level*. Dengan adanya gambar ini, pemain dapat melihat gambar – gambar yang ada sehingga dapat digunakan sebagai sarana untuk belajar kedepannya.

4.2.5 Implementasi *help screen*

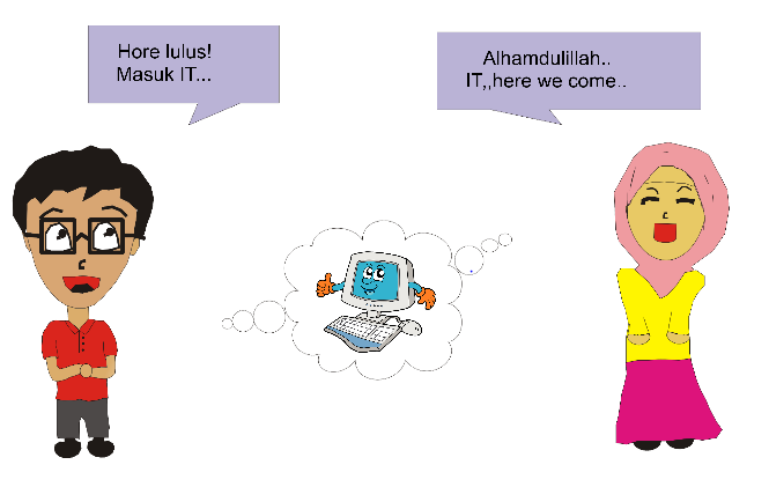
Tampilan halaman *help* memberikan informasi cara bermain dalam permainan algoritma *puzzle*. Pengguna yang kesulitan dalam cara bermain dapat melihat *help screen* setelah masuk menu utama.

4.2.6 Implementasi *credit screen*

Tampilan halaman *credit* memberikan informasi tentang orang-orang yang terlibat dalam pembuatan permainan ini. *Credit screen* ini terdapat di menu utama.

4.2.7 Implementasi Permainan Utama

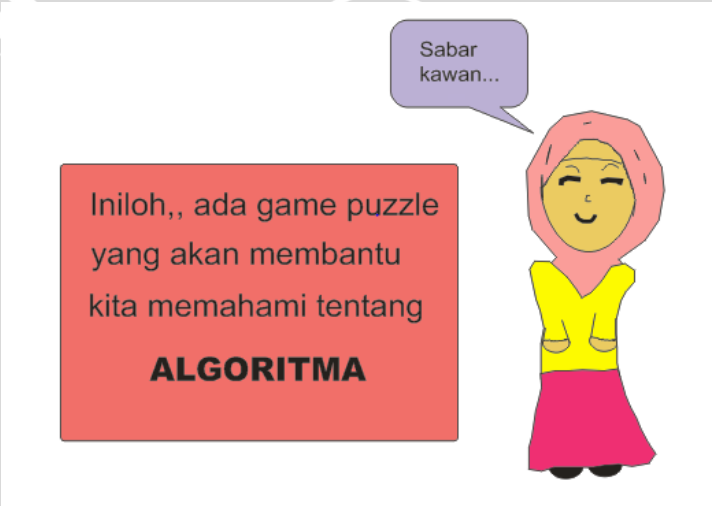
Ketika pemain mengklik tombol start, maka kisah Aleandro dan Aleandra dimulai. Berawal dari kisah Aleandro dan Aleandra yang telah lulus SMA dan diterima di universitas dengan jurusan Teknik Informatika. Mereka sudah membayangkan bahwa kuliah di Informatika dapat menunjang apa yang mereka sukai yaitu berhubungan dengan komputer. *Screen* awal dua remaja tersebut dapat dilihat dalam Gambar 4.3.



Gambar 4. 3 Cerita Awal

Kemudian setelah masa perkuliahan dimulai, Aleandro merasa kesulitan dalam menerima materi mengenai algoritma. Sedangkan algoritma merupakan hal penting yang harus bisa dimengerti mahasiswa Informatika. Aleandra menyemangati temannya tersebut dan memberi solusi bahwa algoritma dapat dipahami dengan menggunakan permainan puzzle. *Screen* tersebut dapat dilihat pada Gambar 4.4 dan Gambar 4.5.

Gambar 4. 4 Cerita Kedua

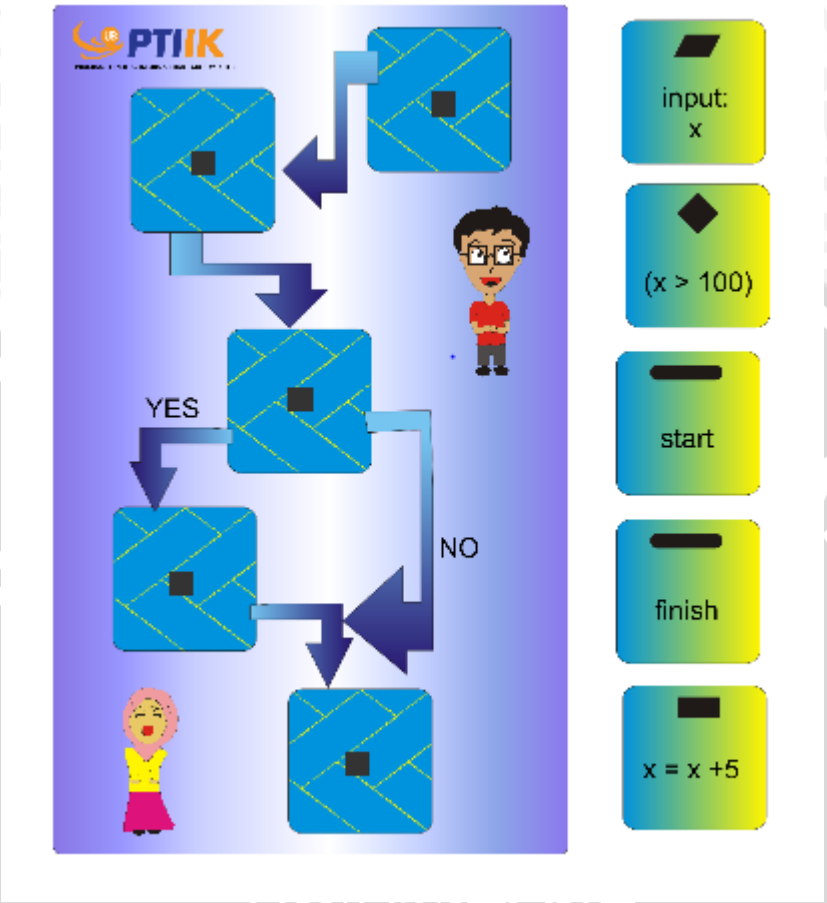


Gambar 4. 5 Cerita Ketiga

Screen berikutnya memberikan gambaran sekilas tentang algoritma. Aleandro pun merasa tertarik untuk mencoba *game puzzle* untuk mempermudah memahami algoritma tersebut agar ia tidak bingung lagi.

Setelah itu akan dilanjutkan dengan screen permainan algoritma *puzzle* yang terdiri dari 9 macam jenis *puzzle* yaitu *flowchart* algoritma instruksi runtunan sederhana, *flowchart* algoritma instruksi runtunan yang lebih panjang, *flowchart* algoritma instruksi pemilihan *if/then/else* bentuk 1 kasus, *flowchart* algoritma instruksi pemilihan *if/then/else* bentuk 2 kasus, *flowchart* algoritma instruksi pemilihan *if/then/else* bentuk bersusun, *flowchart* algoritma instruksi pemilihan *case*, *flowchart* algoritma instruksi perulangan *while-do*, *flowchart*

algoritma instruksi perulangan *repeat-until*, dan *flowchart* algoritma instruksi perulangan *for*. Berikut Gambar 4.6 salah satu contoh permainannya.



Gambar 4. 6 Permainan Algoritma *Puzzle*

4.3 Implementasi Prosedur Program

Aplikasi game pembelajaran algoritma ini memiliki beberapa proses atau method. Beberapa method yang akan dicantumkan dalam penulisan makalah skripsi ini hanya untuk algoritma dari beberapa proses (operasi) utama saja, sehingga tidak semua method akan dicantumkan. Adapun beberapa algoritma tersebut adalah :

4.3.1 Implementasi untuk Mengatur Objek *Puzzle Piece*

Puzzle piece merupakan objek dari *puzzle* yang merupakan kepingan-kepingan dari potongan-potongan algoritma tertentu sesuai *level* yang dibuat. Dalam program diatur agar objek *puzzle* dapat melakukan pergerakan / *drag* yang dikendalikan *mouse* dan melakukan tumbukan yang memerlukan komponen

collider2D dan rigidbody. Diatur juga agar objek puzzle dapat melakukan *snap* dengan *PuzzleSnap*. Akan ditunjukkan pada Tabel 4.3.

Tabel 4. 3 Program Mengatur Objek *Puzzle Piece*

Program *PuzzlePiece*

```
using UnityEngine;
using System.Collections;

public class PuzzlePiece : MonoBehaviour {

      public int ID = -1;
      public bool SnapFlag = false;
//merepresentasikan apakah objek ini lagi "Snap"

      private Vector3 _snap_position; //menyimpan posisi Snap

      private PuzzleManager _puzle_mananger;

      // Use this for initialization
      void Start () {
            _puzle_mananger                                          =
Camera.main.GetComponent<PuzzleManager> ();
      }

      // Update is called once per frame
      void Update () {

      }

      //event ketika ada action mouse drag di atas objek ini
      //event ini memerlukan komponen Collider / Collider2D
      //
      //digunakan agar objek selalu mengikuti mouse ketika didrag
      void OnMouseDrag(){
            Vector3  mouse_pos  =  Camera.main.ScreenToWorldPoint
(Input.mousePosition);
      //convert posisi mouse ke posisi game world
            mouse_pos.z = this.transform.position.z;
      //dilakukan supaya koordinat z game ini masih terjaga
            this.transform.position = mouse_pos;
      }

      //event ketika ada action mouse dilepas di atas objek ini
      //event ini memerlukan komponen Collider / Collider2D
      //
      //pengecekan di sini dilakukan untuk snap objek
      //jika SnapFlag bernilai True, maka objek akan melakukan
snap ke posisi    _snap_position
      void OnMouseUp(){
            if (SnapFlag == true) {
                  this.transform.position = _snap_position;
                  _puzle_mananger.checkGame ();
            }
      }
```

```
      //event ketika terjadi tabrakan / tumbukan dengan game
object lain yang memiliki collider
      //event ini memerlukan komponen Collider / Collider2D &
rigidbody
      //
      //jika objek lain yg sedang bertumbukan dengan objek ini
mempunyai tag="Snap",
      //maka SnapFlag akan bernilai True dan _snap_position akan
menyimpan posisi game objek lain tersebut
      void OnTriggerEnter2D(Collider2D other){
            if (other.tag == "Snap") {

      if(other.GetComponent<PuzzleSnap>().Piece==null||

other.GetComponent<PuzzleSnap>().Piece.gameObject.GetInstanceID
()==this.gameObject.GetInstanceID ()){
                  _snap_position = other.transform.position;
                  SnapFlag=true;
                  }
            }
      }

      //event ketika LEPAS tabrakan / tumbukan dengan game object
lain yang memiliki collider
      //event ini memerlukan komponen Collider / Collider2D &
rigidbody
      //
      //jika objek lain yg sedang bertumbukan dengan objek ini
mempunyai tag="Snap",
      //maka SnapFlag akan bernilai False
      void OnTriggerExit2D(Collider2D other){
            if (other.tag == "Snap") {
                  SnapFlag=false;
            }
      }
}
```

4.3.2 Implementasi untuk Mengatur *Puzzle Snap*

Puzzle snap merupakan objek yang menarik kepingan-kepingan *puzzle* dari potongan-potongan algoritma tertentu sesuai level yang dibuat. Dalam program diatur agar snap dapat melakukan tumbukan yang memerlukan komponen *collider*2D dan *rigidbody*. Diatur juga agar *PuzzleSnap* dapat melakukan *snap* dengan *PuzzlePiece*. Akan ditunjukkan pada Tabel 4.4.

Tabel 4. 4 Program Mengatur *PuzzleSnap*

Program *PuzzleSnap*

`using UnityEngine;`
`using System.Collections;`
 	
`public class PuzzleSnap : MonoBehaviour {`

```
	public int ID=-1;
	public PuzzlePiece Piece;

	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update () {

	}

	//event ketika terjadi tabrakan / tumbukan dengan game
object lain yang memiliki collider
	//event ini memerlukan komponen Collider / Collider2D &
rigidbody
	//
	//jika objek lain yg sedang bertumbukan dengan objek ini
mempunyai tag="Snap",
	//maka SnapFlag akan bernilai True dan _snap_position akan
menyimpan posisi game objek lain tersebut
	void OnTriggerEnter2D(Collider2D other){
		if (other.tag == "Piece" && Piece==null) {
			Piece=other.GetComponent<PuzzlePiece>();
		}
	}
	//event ketika LEPAS tabrakan / tumbukan dengan game object
lain yang memiliki collider
	//event ini memerlukan komponen Collider / Collider2D &
rigidbody
	//
	//jika objek lain yg sedang bertumbukan dengan objek ini
mempunyai tag="Snap",
	//maka SnapFlag akan bernilai False
	void OnTriggerExit2D(Collider2D other){
		if (other.tag == "Piece"&&Piece!=null&&

other.gameObject.GetInstanceID()==Piece.gameObject.GetInstanceID()
) {
			Piece=null;
		}
	}
}
```

4.3.3 Implementasi untuk Mengatur *PuzzleManager*

PuzzleManager ini akan memeriksa apakah ID yang terdapat di *PuzzlePiece* dan *PuzzleSnap* sama. Jika sama maka akan ditambilkan kata MENANG yang menandakan berarti pemain berhasil menyelesaikan permainan. Akan ditampilkan pada Tabel 4.5.

Tabel 4. 5 Program Mengatur *Puzzle Manager*

Program *PuzzleManager*

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

public class PuzzleManager : MonoBehaviour {

	public bool WinGame = false;
	public List<PuzzleSnap> SnapList;

	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update () {

	}

	public void checkGame()
	{
		for(int i=0;i<SnapList.Count;i++)
	//untuk mengecek apakah cocok antara PuzzlePiece dan
PuzzleSnap
		{
			//apabila cocok maka tumbukan dapat diterima dan
terpasang
			if(SnapList[i].ID>=0&&
			   SnapList[i].Piece!=null&&
			   SnapList[i].ID==SnapList[i].Piece.ID)
			{
				WinGame=true;
		//WinGame akan tercentang sebagai tanda semua
PuzzlePiece terpasang tepat pada PuzzleSnap yang telah disediakan
			}
			else
			{
				WinGame=false;
		//WinGame tidak tercentang karena ada yang salah
				break;
			}
		}

		if (WinGame)
			print ("MENANG");
	//mencetak MENANG jika telah berhasil menyelesaikan puzzle
dengan benar
	}
}
```

4.3.4 Implementasi untuk Mengatur Kamera Kontrol

Kamera kontrol merupakan tambahan yang berguna untuk melakukan fungsi-fungsi tertentu pada objek permainan. Dengan *CameraControl* dapat melakukan *zoom in-zoom out* dan menggerakkan semua *game* objek yang ada termasuk *PuzzlePiece* dan *PuzzleSnap*. Akan ditunjukkan pada Tabel 4.6.

Tabel 4. 6 Program Mengatur Kamera Kontrol

Program *CameraControl*

```
using UnityEngine;
using System.Collections;

public class CameraControl : MonoBehaviour {

      public Transform Game_Object;
      public Vector2 MaxPos;

      public float MinOrthoValue=2;
      public float MaxOrthoValue=25;
      public float ZoomModifier=2.0f;

      private Vector3 _start_mouse;
      private Vector3 _start_position;


      void Awake () {
            MaxPos.y = this.camera.orthographicSize;
            MaxPos.x      =      this.camera.orthographicSize      *
this.camera.aspect;
      }

      // Update is called once per frame
      void Update () {

            float    mouseScroll     =     Input.GetAxis     ("Mouse
ScrollWheel");
            if (mouseScroll != 0) {
                  float  new_zoom  =  this.camera.orthographicSize-
(mouseScroll*ZoomModifier);
                  new_zoom=Mathf.Clamp
(new_zoom,MinOrthoValue,MaxOrthoValue);
                  this.camera.orthographicSize=new_zoom;
            }

            if (Input.GetMouseButtonDown (1)) {
                  _start_mouse   =   this.camera.ScreenToWorldPoint
(Input.mousePosition);
                  _start_position=Game_Object.position;
            }

            if (Input.GetMouseButton (1)) {
                  Vector3
delta_mouse=this.camera.ScreenToWorldPoint  (Input.mousePosition)-
```

```
_start_mouse;
                delta_mouse.z=Game_Object.transform.position.z;

                Vector3 new_pos=_start_position+delta_mouse;
                new_pos.y=Mathf.Clamp                (new_pos.y,-
MaxPos.y,MaxPos.y);
                new_pos.x=Mathf.Clamp                (new_pos.x,-
MaxPos.x,MaxPos.x);

                Game_Object.transform.position=new_pos;
            }

        }
}
```

BAB V
PENGUJIAN DAN ANALISIS

Bab ini membahas mengenai tahapan pengujian dan analisis permainan *Puzzle* Algoritma yang telah dibangun. Proses pengujian dilakukan melalui dua tahapan yaitu pengujian unit dan pengujian integrasi. Pengujian unit dan integrasi menggunakan teknik pengujian *White Box* (*White Box Testing*).

5.1 Pengujian Unit

Pada pengujian unit digunakan *White Box Testing* dengan teknik *Basis Path Testing*. Pada teknik *Basis Path Testing* proses pengujian dilakukan dengan memodelkan program pada sebuah *flow graph* dan *cyclometic complexity*.

5.1.1 Pengujian Unit untuk PuzzlePiece

Puzzle piece merupakan komponen penting dalam permainan ini untuk mengatur gerakan objek *piece puzzle*, tumbukan antar objek dan pemberian ID untuk melakukan *snap* dengan *PuzzleSnap*. Pemodelan *PuzzlePiece* dalam bentuk *flowgraph* ditunjukkan pada Tabel 5.1. *Flowgraph* pengujian ditunjukkan pada Gambar 5.1.

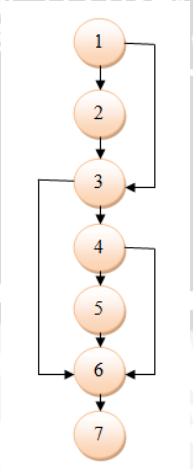
Tabel 5. 1 Pemodelan *flowgraph* program *PuzzlePiece*

Program *PuzzlePiece*

```
using UnityEngine;
using System.Collections;

public class PuzzlePiece : MonoBehaviour {

	public int ID = -1;
	public bool SnapFlag = false;
//merepresentasikan apakah objek ini lagi "Snap"

	private Vector3 _snap_position; //menyimpan posisi Snap

	private PuzzleManager _puzle_mananger;

	// Use this for initialization
	void Start () {
		_puzle_mananger                                    =
Camera.main.GetComponent<PuzzleManager> ();
	}

	void Update () {
```

```
    }


    void OnMouseDrag(){
    Vector3                              mouse_pos                    =
Camera.main.ScreenToWorldPoint(Input.mousePosition);
    mouse_pos.z = this.transform.position.z;
        this.transform.position = mouse_pos;
    }

    void OnMouseUp(){
        if (SnapFlag == true) {
            this.transform.position = _snap_position;
            _puzle_mananger.checkGame ();
        }
    }

    void OnTriggerEnter2D(Collider2D other){
        if (other.tag == "Snap") {

    if(other.GetComponent<PuzzleSnap>().Piece==null||

other.GetComponent<PuzzleSnap>().Piece.gameObject.GetInstanceID
()==this.gameObject.GetInstanceID ()){
            _snap_position = other.transform.position;
            SnapFlag=true;
            }
        }
    }

    void OnTriggerExit2D(Collider2D other){
        if (other.tag == "Snap") {
            SnapFlag=false;
        }
    }
}
```

1 2 3 4 5 6 7



Gambar 5. 1 *flowgraph* pengujian *PuzzlePiece*

Pemodelan ke dalam *flowgraph* yang telah dilakukan untuk menguji *PuzzlePiece* menghasilkan jumlah kompleksitas siklomatis (*cyclomatic complexity*) melalui persamaan V(G) = E – N + 2, dimana V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi atau *edge* (garis penghubung antar *node*) dan N merupakan jumlah simpul (*node*).

V(G) = E – N + 2
= 9 – 7 + 2
= 4

Berdasarkan dari nilai *cyclomatic complexity* yang telah didapatkan dari perhitungan maka ditentukan empat buah basis set dari jalur *independent*, yaitu :

Jalur 1 : 1 – 3 – 4 – 5 – 6 – 7
Jalur 2 : 1 – 2 – 3 – 6 – 7
Jalur 3 : 1 – 2 – 3 – 4 – 6 – 7
Jalur 4 : 1 – 2 – 5 – 6 – 7

5.1.2 Pengujian Unit untuk *PuzzleSnap*

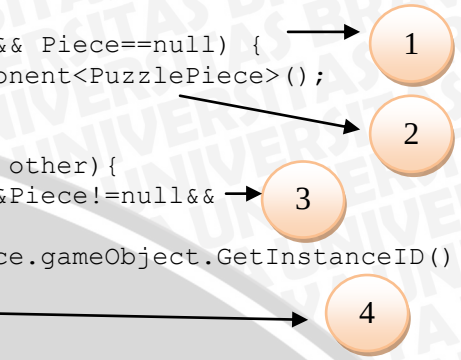
PuzzleSnap mengatur untuk menangkap *PuzzlePiece*, sehingga satu sama lain dapat terpasang menjadi *puzzle* yang utuh dan benar. Pada *PuzzleSnap* diberikan ID yang akan dicocokkan dengan ID *PuzzlePiece* sehingga dapat melakukan *snap* satu sama lain. Permodelan *PuzzleSnap* dalam bentuk *flowgraph* ditunjukkan pada Tabel 5.2. *Flowgraph* pengujian ditunjukkan pada Gambar 5.2.

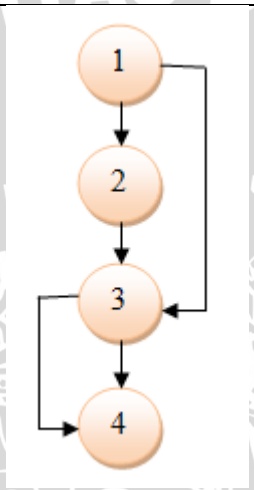
Tabel 5. 2 Permodelan *flowgraph* program *PuzzleSnap*

Program *PuzzleSnap*

`using UnityEngine;`
`using System.Collections;`
 	
`public class PuzzleSnap : MonoBehaviour {`
 	
`public int ID=-1;`
`public PuzzlePiece Piece;`
 	
`// Use this for initialization`
`void Start () {`
 	
`}`
 	
`// Update is called once per frame`
`void Update () {`

```
    }
    void OnTriggerEnter2D(Collider2D other){

        if (other.tag == "Piece" && Piece==null) {
            Piece=other.GetComponent<PuzzlePiece>();
        }
    }
    void OnTriggerExit2D(Collider2D other){
        if (other.tag == "Piece"&&Piece!=null&&
other.gameObject.GetInstanceID()==Piece.gameObject.GetInstanceID()
) {
            Piece=null;
        }
    }
}
```





Gambar 5. 2 *flowgraph* pengujian *PuzzleSnap*

Pemodelan ke dalam *flowgraph* yang telah dilakukan untuk menguji *PuzzleSnap* menghasilkan jumlah kompleksitas siklomatis (*cyclomatic complexity*) melalui persamaan V(G) = E – N + 2, dimana V(G) merupakan jumlah kompleksitas siklomatis, E merupakan sisi atau *edge* (garis penghubung antar *node*) dan N merupakan jumlah simpul (*node*).

V(G) = E – N + 2

= 5 – 4 + 2

= 3

Berdasarkan dari nilai *cyclomatic complexity* yang telah didapatkan dari perhitungan maka ditentukan tiga buah basis set dari jalur *independent*, yaitu :

Jalur 1 : 1 – 3 – 4

Jalur 2 : 1 – 2 – 3

Jalur 3 : 1 – 2 – 3 – 4

5.2 Analisa Pengujian Unit

Hasil analisis yang didapatkan dari pengujian unit yaitu berdasarkan perhitungan *cyclomatic complexity* dari tiap *flowgraph* kode unit, kode unit yang menghasilkan jalur kasus uji terbanyak adalah kode operasi untuk menghasilkan nilai acak sebanyak empat kasus uji. Jumlah kasus uji kode ini sebagian besar dipengaruhi dari seleksi kondisi.

5.3 Pengujian Integrasi

Pengujian integrasi diterapkan pada *method* yang mengintegrasikan fungsionalitas dari *class-class* lain untuk melakukan sebuah operasi tertentu. Pada pengujian integrasi perangkat lunak ini digunakan teknik *White-Box Testing* dengan teknik *Basis path Testing*. Pada teknik *Basis Path testing*, proses pengujian dilakukan dengan memodelkan algoritma pada suatu *flow graph*, menentukan jumlah kompleksitas siklomatis (*cyclomatic complexity*) dan menentukan sebuah basis set dari jalur independen.

5.3.1 Pengujian Integrasi *PuzzleManager*

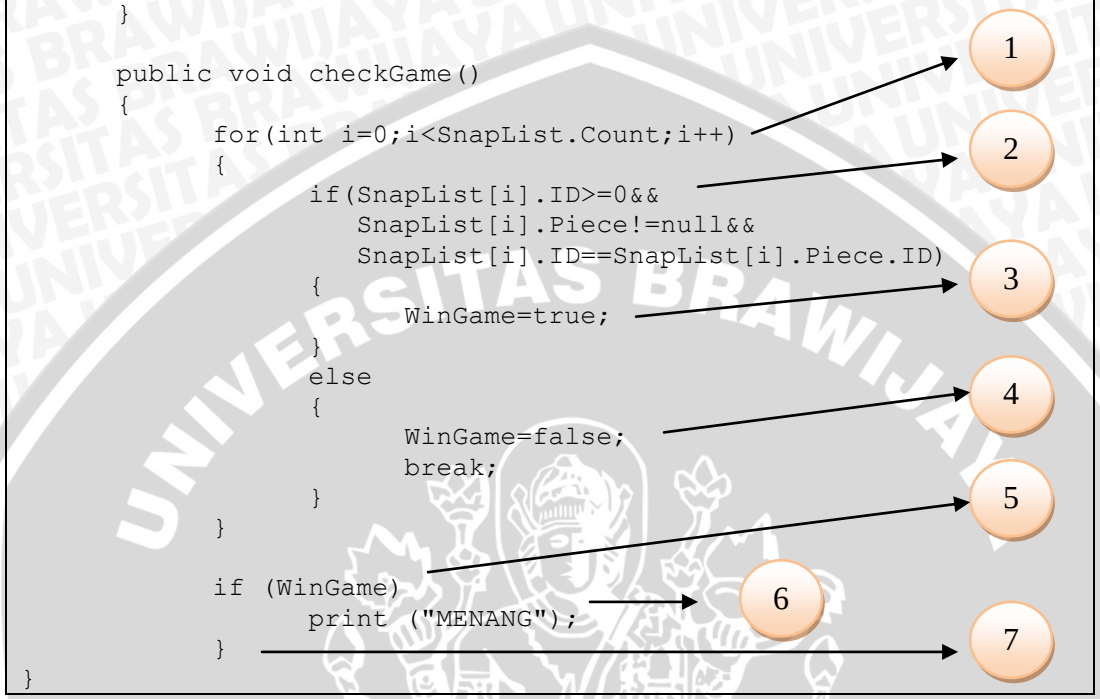
Pengujian pada *PuzzleManager* ini akan memeriksa apakah antara *PuzzlePiece* dan *PuzzleSnap* dapat saling bertumbukan dan ID memiliki kesamaan sehingga dapat saling melakukan snap. Pemodelan *PuzzleManager* dalam bentuk *flowgraph* ditunjukkan pada Tabel 5.3. *Flowgraph* pengujian ditunjukkan pada Gambar 5.3.

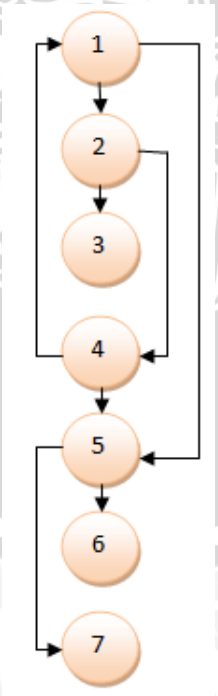
Tabel 5. 3 Pemodelan *PuzzleManager*

Program *PuzzleManager*

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

public class PuzzleManager : MonoBehaviour {

	public bool WinGame = false;
	public List<PuzzleSnap> SnapList;

	// Use this for initialization
	void Start () {
```

```
    }

    // Update is called once per frame
    void Update () {

    }

    public void checkGame()
    {
        for(int i=0;i<SnapList.Count;i++)
        {
            if(SnapList[i].ID>=0&&
               SnapList[i].Piece!=null&&
               SnapList[i].ID==SnapList[i].Piece.ID)
            {
                WinGame=true;
            }
            else
            {
                WinGame=false;
                break;
            }
        }

        if (WinGame)
            print ("MENANG");
        }
}
```





Gambar 5. 3 *flowgraph PuzzleManager*

5.4 Analisa Pengujian Integrasi

Proses analisis terhadap hasil pengujian integrasi dilakukan dengan melihat kesesuaian unit modul yang menyusun satu blok fungsi dalam permainan. Dari seluruh pengujian integrasi yang dilakukan, jumlah jalur pada logika setiap prosedur telah sesuai dengan perhitungan *cyclomatic complexity*. Setiap kasus uji yang dibuat berdasarkan jumlah jalur tersebut telah diuji dan memberikan hasil sesuai dengan hasil yang diharapkan. Berdasarkan hal tersebut, maka dapat diambil kesimpulan bahwa unit modul dari program sudah memenuhi kebutuhan fungsional yang telah dirancang pada tahap perancangan.

5.5 Pengujian terhadap Pengguna

Pengujian terhadap pengguna dilakukan dengan cara memberikan kuisioner kepada responden. Penulis mendapatkan data dari responden mahasiswa informatika/ilmu komputer dengan kisaran usia 19 – 22 tahun. Penulis menghampiri beberapa mahasiswa dan menanyakan seberapa besar pemahaman mahasiswa informatika/ilmu komputer mengenai algoritma. Setelah mendapatkan jawaban dari pengguna, penulis memberikan kesempatan kepada pengguna untuk mencoba memainkan permainan algoritma *puzzle*. Setelah pengguna selesai memainkan, penulis mengambil data tentang pendapat dari pengguna serta meminta pengguna mengisi kuisioner yang terdapat di lampiran L-1. Hasil pengujian kuisioner terdapat pada lampiran L-2 dan L-3.

5.6 Analisa Pengujian Pengguna

Kuisioner diberikan kepada keseluruhan total responden sebanyak tiga puluh orang. Dari tiga puluh responden mahasiswa berusia antara 19 – 22 tahun. Adapun hasil yang didapat sesuai dengan kuisioner terdapat pada lampiran L-2 sedangkan analisis dari kuisioner tersebut adalah sebagai berikut :

1. Dari pertanyaan kuisioner yang pertama tentang pentingnya memahami algoritma dalam informatika/ilmu komputer didapat bahwa 100% responden mengatakan penting dan tidak ada yang mengatakan tidak penting. Analisa terhadap pengguna membuktikan bahwa pentingnya

untuk memahami algoritma dalam perkuliahan di informatika/ilmu komputer.

2. Dari pertanyaan kuisioner yang kedua tentang pemahaman pengguna tentang algoritma didapat bahwa 30% mengatakan memahami algoritma, 70% responden kurang memahami algoritma dan tidak ada yang mengatakan tidak memahami algoritma. Analisa terhadap pengguna bahwa sebagian mahasiswa informatika/ilmu komputer masih kurang memahami tentang algoritma.
3. Dari pertanyaan kuisioner yang ketiga tentang kemampuan mahasiswa informatika/ilmu komputer dalam menyusun algoritma sederhana didapat bahwa 66,67% responden mengatakan dapat menyusun algoritma sederhana, 33,33% kurang dapat menyusun algoritma sederhana, dan tidak ada yang mengatakan tidak mampu menyusun algoritma sederhana.
4. Dari pernyataan kuisioner keempat bahwa algoritma dapat melatih pola pikir secara sistematis/terurut didapat 100% responden yang setuju dengan pernyataan tersebut dan tidak ada yang tidak setuju dengan pernyataaan tersebut.
5. Dari pertanyaan kuisioner yang kelima tentang apakah dibutuhkan latihan untuk mengasah dalam memahami algoritma didapat bahwa 100% responden mengatakan perlu latihan tersebut dan tidak ada yang mengatakan tidak perlu.
6. Dari pertanyaan kuisioner yang keenam tentang apakah dibutuhkan aplikasi yang bertujuan untuk membuat pola pikir menjadi sistematis didapat bahwa 80% mengatakan perlu adanya aplikasi untuk melatih pola pikir menjadi sistematis dan 20% mengatakan tidak perlu adanya aplikasi tersebut.
7. Dari pertanyaan kuisioner yang ketujuh tentang implementasi algoritma dalam sebuah permainan/*game* didapat bahwa 100% responden setuju dengan adanya implementasi tersebut dan tidak ada responden yang mengatakan tidak setuju. Analisa terhadap pengguna bahwa implementasi algoritma dalam permainan/*game* dapat diterapkan untuk latihan dalam mempelajari algoritma.

8. Dari pertanyaan kuisioner yang kedelapan tentang ketertarikan mahasiswa untuk belajar memahami algoritma melalui *game puzzle* didapat bahwa 83,33% responden tertarik melalui model pembelajaran melalui *game*, 13,33% kurang tertarik dan 3,33% tidak tertarik dengan *game*. Analisa terhadap pengguna bahwa melalui pembelajaran permainan edukasi dapat lebih menarik minat mahasiswa informatika/ilmu komputer dalam mempelajari algoritma.
9. Dari pertanyaan kuisioner yang kesembilan tentang apakah mahasiswa dapat lebih mengerti dan berpikir sistematis dengan memainkan game algoritma puzzle didapat bahwa 90% dapat mengerti, 10% kurang mengerti dan tidak ada yang mengatakan tidak mengerti.
10. Dari pertanyaan kuisioner yang kesepuluh tentang konsep cara belajar sambil bermain game seperti contoh game yang dibuat didapat bahwa 100% responden menyatakan menyukai konsep tersebut dan tiak ada yang menyatakan kurang/tidak suka.

Dari keseluruhan pengujian dan analisa terhadap pengguna didapat kesimpulan bahwa aplikasi ini telah memenuhi kebutuhan dan sesuai dengan tujuan yaitu melatih mahasiswa informatika/ilmu komputer untuk memahami algoritma. Hal ini dibuktikan dengan jawaban pada kuisioner nomor 8 dan nomor 9, para pengguna menjawab bahwa mereka tertarik dan lebih mengerti tentang algoritma dengan permainan ini.

Dari pengujian mendalam mengenai seberapa besar kemampuan mahasiswa informatika / ilmu komputer dalam menyusun algoritma didapat hasil bahwa kesulitan yang dialami berupa penyusunan bahasa yang harus digunakan dan bagaimana langkah – langkah penyelesaian algoritma yang harus dilakukan.

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Algoritma puzzle adalah sebuah permainan edukasi yang membantu pemain khususnya mahasiswa informatika untuk melatih dalam memahami cara kerja algoritma yang direpresentasikan dalam flowchart yang sudah dimodifikasi. Berdasarkan hasil pengamatan selama perancangan, implementasi, dan proses pengujian permainan yang dilakukan, maka diambil kesimpulan sebagai berikut :

1. Memahami algoritma dapat dilakukan dengan pembelajaran simulasi yang diimplementasikan pada suatu *game* edukasi untuk melatih pola pikir berjalan secara terurut. Dengan *game*, maka pelajaran di bidang informatika seperti algoritma dapat lebih mudah dipahami dengan konsep belajar yang lebih menyenangkan karena mahasiswa dapat bermain sekaligus belajar.
2. Permainan *game puzzle* untuk memudahkan memahami cara kerja algoritma ini diimplementasikan dengan menggunakan *Unity* karena merupakan *game engine* yang *user friendly* dan mudah penerapannya. Perancangan dilakukan dengan representasi berupa gambar flowchart modifikasi untuk menarik pengguna.
3. Pengujian unit permainan edukasi "Algoritma *Puzzle*" dengan metode *White Box* dilakukan pada unit *PuzzlePiece* dan *PuzzleSnap* serta pada *PuzzleManager*. Pengujian unit *PuzzlePiece* menghasilkan jumlah kompleksitas siklomatis sebanyak empat dan sesuai dengan output kode unit yang diharapkan dengan menghasilkan jalur kasus uji terbanyak pada kode operasi yaitu menghasilkan nilai acak sebanyak empat kasus uji. Pengujian unit *PuzzleSnap* dan PuzzleManager menghasilkan jumlah kompleksitas siklomatis sebanyak tiga, sesuai dengan output kode unit yang diharapkan dengan menghasilkan jalur kasus uji pada kode operasi yaitu nilai acak sebanyak tiga kasus uji. Pengujian terhadap pengguna melalui kuisioner dan analisis mendalam melalui studi literatur,

menunjukkan bahwa *game* sudah sesuai dengan kebutuhan dan memenuhi tujuan.

6.2 Saran

Saran untuk pengembangan permainan "Algoritma *Puzzle*" lebih lanjut antara lain :

1. Dapat dilakukan pengembangan dengan membuat versi *mobile*, *tablet* dan sebagainya. Aplikasi ini dikembangkan hanya untuk perangkat PC. Dengan adanya pengembangan dapat membuat pengguna bisa memainkan dimana saja melalui *platform* lain.
2. Penambahan cakupan materi algoritma tidak terbatas pada instruksi utama saja. Dapat pula dikembangkan dengan representasi lain seperti *pseudocode*.
3. Penambahan jumlah level yang lebih sulit lagi. Penambahan level bisa disesuaikan dengan tingkat kesulitan *flowchart*.

DAFTAR PUSTAKA

[GRI-09] Griffith, Christopher. 2009. *Real World Flash Game Development. Focal Press*. USA.

[KAD-05] Kadir, Abdul & Heriyanto. 2005. *Algoritma Pemrograman Menggunakan C++*. Penerbit ANDI

[MUT-08] Mutaqin,Didih Haryanto. 2008. *Upaya peningkatan Pencapaian Hasil Belajar Mahasiswa Mata Kuliah Pemrograman Komputer Melalui Pendekatan Pembelajaran Kooperatif denagn Memanfaatkan Media Pembelajaran Interaktif*. Skripsi. Yogyakarta.

[NUG-12] Nugraha, Ngurah Wira., Pradita, I Putu Gede Dharma., Hartawan, Yudi., Apsari, Putu Jea Mitha. 2012. *Enhanced Intelligent Puzzle (Enizle) Sebagai Media Permainan Edukatif Bagi Penyandang Tuna Netra*. PKM Institut Teknologi Telkom. Bandung.

[PRE-10] Pressman, Roger. 2010. *Software Engineering : A Practioner's Approach, 7th Edition*. Mc Graw-Hill.

[ROG-10] Roger, Scott. 2010. *Level Up! The Guide to Great Video Game Design*. Wiley : West Sussex.

[SAP-11] Saputra, Puput Windi. 2011. *Rancang Bangun Game RPG Sejarah Kerajaan-Kerajaan Di Indonesia Dengan Pembelajaran Bahasa Jawa Untuk Anak-Anak dan Remaja*. Tugas Akhir. Surabaya.

[SEM-09] Sembiring, Yosua Yudhanata. 2009. *Algoritma dan Implementasi Alat Bantu Pemecahan Masalah Matematika*. Skripsi. Medan.

[SUA-12] Suarga, Dr., M. Sc., M.Math.,Ph.D. 2006. *Algoritma dan Pemrograman*. Andi. Yogyakarta.

[SUT-04] Sutedjo, Budi S. Kom. dan Michael AN, S.Kom., MM. 2004. *Algoritma & Teknik Pemrograman Konsep, Implementasi, dan Aplikasi*. Andi. Yogyakarta.

[TAR-12] Tarigan, Ammahli Fakar. 2012. *Analisis Dan Implementasi Algoritma Linear Search pada Permainan Word Scramble*. [pdf], repository.usu.ac.id diakses pada 14 November 2013

[ZAI-10] Zaida, Efrizal. 2010. *Pengantar Desain dan Analisis Algoritma, Edisi 2. Buku 1*. Penerbit Salemba Infotek