WANDAH WIBAWANTO

DASAR PENGEMBANGAN GAME HTML 5

Penerbit LPPM
Universitas Negeri Semarang

[halaman ini sengaja dikosongkan]

DASAR PENGEMBANGAN *GAME* HTML 5

Wandah Wibawanto

Penerbit LPPM
Universitas Negeri Semarang

DASAR PENGEMBANGAN *GAME* HTML 5

Penulis:
Wandah Wibawanto, S.Sn. M.Ds.

ISBN : 978-623-366-004-4

No Pencatatan Hak Cipta :
000264942

Desain & Tata Letak:
Wandah Wibawanto, S.Sn. M.Ds.

Penerbit :
Penerbit LPPM Universitas Negeri Semarang
Gedung Prof. Dr. Retno Sriningsih Satmoko, Penelitian dan Pengabdian Masyarakat,
Kampus Sekaran, Gunungpati, Semarang 50229

Email sentraki@mail.unnes.ac.id

Cetakan Pertama : 2021 (PDF)

Distributor:
Penerbit LPPM Universitas Negeri Semarang

Kata Pengantar

Game merupakan salah satu sub sektor ekonomi kreatif yang selalu mengalami peningkatan dalam hal pendapatan, jumlah pemain dan teknologi. Teknologi dalam pengembangan *game* juga mengalami perkembangan yang sangat signifikan, *game-engine* maupun *authoring tool* selalu mengalami peningkatan guna memberikan kemudahan bagi pengembang game. Salah satu teknologi dalam pengembangan *game* adalah HTML 5 *canvas* dengan bahasa pemrograman *Javascript*.

Dalam 2 dekade terakhir, *web browser* menampilkan interaktifitas dalam bentuk aplikasi maupun *game* melalui teknologi *Flash Player*. Namun, seiring berhentinya layanan *Flash Player* di tahun 2021, pengembangan aplikasi atau *game* untuk *web browser* berpindah ke teknologi HTML 5 dengan fitur barunya yaitu *canvas*. Pengembangan *game* HTML5 dapat dilakukan dengan berbagai aplikasi seperti Unity, Construct, GDevelop atau menggunakan *framework* seperti Phaser, MelonJS, PlayJS dan sebagainya. Kendati demikian pengenalan dasar pengembangan *game* mutlak diperlukan bagi seseorang yang mulai belajar pengembangan *game* HTML5.

Pada saat ini tutorial lebih mudah didapatkan melalui video pada kanal-kanal media sosial seperti Facebook atau YouTube, akan tetapi tetap diperlukan sebuah literatur tertulis yang membahas secara spesifik dasar pengembangan *game* HTML 5. Minimnya literatur yang membahas baik secara teori maupun secara praktik tentang proses desain sekaligus proses pemrograman *game* HTML5 mengakibatkan kurang berkembangnya *game* HTML5 yang dibuat oleh developer lokal. Padahal *game* HTML5 memiliki pasar khusus yang masih terbuka lebar.

Sebagai respon dari permasalahan tersebut, melalui buku ini penulis berusaha untuk membahas tentang proses desain, prinsip desain *game* HTML 5 sampai dengan proses pemrograman dengan aplikasi yang mudah didapatkan dan gratis. Pembahasan dalam bentuk teori dan praktik, dilengkapi dengan penjelasan ditiap-tiap program, dengan langkah-langkah yang paling mudah untuk diikuti oleh semua kalangan. Semoga buku ini membawa kebermanfaatan bagi perkembangan *game* tanah air, dan mohon maaf sebesar-besarnya apabila banyak kekurangan yang ada dalam buku ini.

Penulis,

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

HTML 5 CANVAS

menawarkan opsi
pengembangan web game
yang dinamis dan semakin mature

BAB 1

PENGENALAN DASAR HTML *CANVAS*

1.1 HTML *Canvas*

Dalam membuat sebuah *game* atau aplikasi interaktif, kita dapat memanfaatkan beberapa *software* atau bahasa pemrograman yang mendukung operasi grafik dan interaktivitas. Beberapa *software* yang cukup populer untuk membuat simulasi antar lain Adobe Flash/Adobe Animate, Smart Apps Creator, Unity3D, Unreal Engine dan beberapa lainnya, sedangkan bahasa pemrograman yang secara umum digunakan untuk membangun sebuah simulasi pada umumnya adalah bahasa C, *Javascript*, HTML, *Actionscript* dan beberapa lainnya. Pada saat buku ini ditulis terdapat beberapa *game* engine populer seperti Unity 3D dan Unreal Engine, sementara untuk membuat *game* atau aplikasi berbasis HTML 5 terdapat *framework* populer seperti Phaser, Babylon JS, atau aplikasi *authoring tools* seperti Construct, Gdevelop, Play*Canvas* dan sebagainya (Faas, 2017).

HTML 5 adalah sebuah bahasa pemrograman untuk menstrukturkan dan menampilkan sebuah tampilan web (*World Wide Web*), yang merupakan elemen utama dari internet. HTML 5 merupakan revisi kelima dari HTML yang mana HTML mulai dikembangkan pada tahun 1990. Tujuan utama pengembangan HTML5 adalah untuk memperbaiki teknologi HTML agar mendukung teknologi multimedia terbaru, mudah dibaca oleh manusia dan juga mudah dimengerti oleh mesin. Salah satu fitur yang didukung oleh HTML 5 yang dapat dimanfaatkan untuk memanipulasi grafis dan membentuk interaktivitas adalah `<canvas>`.

Canvas merupakan salah satu elemen dari HTML 5 yang memungkinkan untuk pengelolaan grafis dinamis menggunakan kode. *Canvas* divisualisasikan dalam bentuk area yang memiliki atribut lebar (*width*) dan tinggi (*height*), dapat dimanipulasi dengan cepat menggunakan bahasa pemrograman *Javascript*. Sebagaimana dengan kanvas kosong tempat pelukis bekerja, elemen `<canvas>` menyediakan area berbasis *bitmap* kepada pengguna untuk menggambar grafik menggunakan *Javascript*. Area tersebut dapat digunakan untuk menggambar grafik, membuat komposisi foto, membuat animasi 2D dan 3D (digunakan dengan WebGL), melakukan pemrosesan atau *rendering* video *real-time* dan dapat dijalankan di sebagian besar aplikasi *internet browser*.

1.2 Struktur HTML *Canvas*

Sebelum pembahasan tentang *canvas*, kita perlu mengenal sedikit tentang standar HTML5 yang akan kita gunakan untuk membuat aplikasi/game. HTML adalah bahasa standar yang digunakan untuk membuat halaman di *World Wide Web*. Halaman HTML dasar dibagi menjadi beberapa bagian, umumnya `<head>` dan `<body>` yang seringkali disebut sebagai *tag*. Spesifikasi HTML5 baru menambahkan *tag* baru beberapa, seperti `<nav>`, `<article>`, `<header>`, dan `<footer>`.

Tag `<head>` biasanya berisi informasi yang akan digunakan oleh HTML seperti judul halaman, *meta data*, dan kode-kode tambahan. Sedangkan *tag* `<body>` untuk membuat halaman konten halaman HTML. Secara sederhana struktur kode HTML adalah sebagai berikut :

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.        <title>Judul Halaman</title>
5.  </head>
6.  <body>
7.        <p>Anda sedang belajar HTML5</p>
8.  </body>
9.  </html>
```

Kode di atas dapat dijalankan oleh internet *browser* dan akan menampilkan konten yang terdapat di antara *tag* `<body>`, yaitu tulisan "Anda sedang belajar HTML5".

Tag `<canvas>` merupakan fitur yang pertamakali dikenalkan oleh Apple pada tahun 2004, dan seiring waktu menjadi standar bagi *internet browser*. Seperti penjelasan sebelumnya *tag* `<canvas>` akan menjadikan konten halaman web dapat dimanipulasi dengan mengelola grafik dan interaktivitas. *Tag* `<canvas>` diletakkan di antara *tag* `<body>`. Pada tutorial yang terdapat di buku ini tag `<canvas>` diletakkan di dalam tag `<div id="gameArea">` untuk mempermudah proses pengaturan tampilan aplikasi pada *web browser* nantinya. Secara menyeluruh, struktur kode HTML yang akan digunakan di dalam tutorial di buku ini adalah sebagai berikut :

```
1. <html>
2. <head>
3.     <meta charset="UTF-8">
4.     <title>Judul Aplikasi</title>
5.     <style>
6. html,body { margin: 0;overflow: hidden;}
7. canvas{ padding: 0; margin: auto; display: block; width:100%;}
```

```
8.  #gameArea {width:80%; position:absolute;}
9.  #game {margin:50px;}
10.     </style>
11.     <script type="text/javascript"
                    src="http://www.wandah.org/js/gameLib.js"></script>
12. </head>
13. <body>
14.      <div id="game">
15.          <div id="gameArea"><canvas id="canvas"></canvas></div>
16.       </div>
17. </body>
18. <script type="text/javascript" src="js/game.js"></script>
19. </html>
```

Interaktivitas yang ditampilkan dalam *canvas* HTML diatur oleh kode *Javascript*. *Javascript* merupakan bahasa kode atau pemrograman yang digunakan untuk menciptakan dan mengendalikan konten website agar menjadi lebih dinamis (Ramtal dkk, 2014) . Konten web dapat bergerak atau berubah tanpa mengharuskan pengguna memuat ulang laman situs secara manual. Kode *Javascript* tidak membutuhkan *compiler* karena *web browser* mampu menginterpretasikannya bersamaan dengan kode HTML. *Javascript* bisa terletak di dalam laman suatu situs atau berada pada *file* .js yang terpisah.

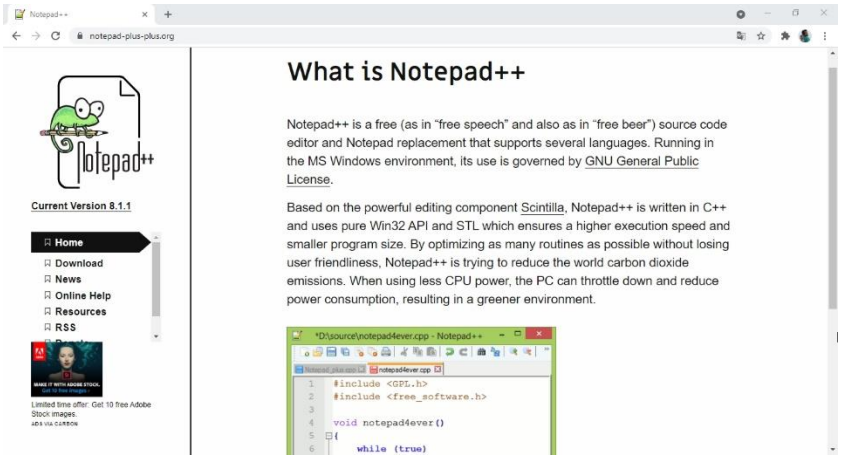
Pada baris 11 digunakan kode **gameLib.js** yang akan diunduh dari situs www.wandah.org. *File* **gameLib.js** ini merupakan kumpulan kode yang digunakan untuk menyederhanakan proses pengembangan aplikasi, yang akan dibahas secara mendetail pada bab-bab selanjutnya. Sedangkan pada baris 18 digunakan kode game.**js** yang merupakan kode *Javascript* yang akan digunakan sebagai kode utama dalam membangun aplikasi/game.

1.3 Perangkat dalam Mengembangkan Aplikasi berbasis HTML *Canvas*

HTML5 dengan fitur *canvas* dapat dijalankan di semua *web browser* terbaru. Kelebihan ini mengisyaratkan bahwa untuk mengembangkan aplikasi berbasis HTML5 *canvas*, kita hanya membutuhkan sebuah *web browser* (yang secara umum telah terinstall di masing-masing komputer), serta sebuah teks editor untuk mengedit teks konten HTML5 ataupun *Javascript*.

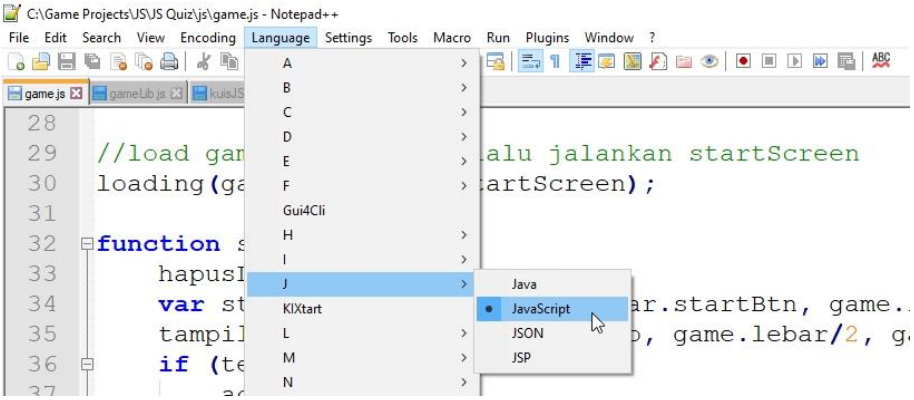
Untuk mengedit teks HTML maupun kode *Javascript* kita dapat menggunakan aplikasi editor teks sederhana seperti Notepad, namun penulis merekomendasikan aplikasi Notepad++ yang dapat diunduh secara gratis pada situs https://notepad-plus-plus.org/ . Notepad++ memiliki keunggulan dibandingkan dengan Notepad standar, yaitu memiliki struktur baris angka

pengkodean dan mendukung beberapa bahasa pemrograman (salah satunya adalah *Javascript*), serta sangat ringan untuk dijalankan di komputer.



Gambar 1. Tampilan halaman notepad-plus-plus

Anda dapat mengunduh *file* Notepad++ secara gratis dan menginstallnya di komputer anda. Dalam pengetikan kode *Javascript* nantinya, anda dapat mengatur jenis bahasa pemrograman yang anda gunakan melalui menu **Langguage > J > *Javascript***, hal ini akan mempermudah proses penulisan, karena kode akan ditampilkan dalam warna yang berbeda, sehingga kita dapat mengetahui struktur kode dan penulisan yang tepat.



Gambar 2. Opsi bahasa pada Notepad++

Pada dasarnya setelah kode HTML dan kode *Javascript* telah dituliskan, maka untuk menjalankan aplikasi cukup digunakan *web browser* seperi Google Chrome, Mozilla Firefox, Safari atau Internet Explorer. Pada tahapan selanjutnya, *file* tersebut juga dapat diunggah ke *server* dan aplikasi secara langsung akan dapat dijalankan pada halaman website tempat *file* diunggah. Dengan kata lain, pengembangan aplikasi berbasis HTML 5 *Canvas* dan *Javascript* tidak membutuhkan *software* komersial apapun, dapat dikembangkan dengan mudah dan tidak membutuhkan komputer dengan spesifikasi tinggi.

1.4 Kerangka Kerja (*Frame Work*) dan Perpustakaan Kode (*Library*) *Javascript*

Pengembangan *Game* atau aplikasi menggunakan *Javascript* menjadi hal yang umum pada 1 dekade terakhir. *Javascript* berkembang menjadi bahasa pemrograman yang fleksibel yang beberapa tahun terakhir, *browser* dan perangkat seluler telah menggabungkan fitur seperti *WebGL* untuk meningkatkan kemampuan grafisnya. Perkembangan ini juga didukung oleh adanya *framework* (kerangka kerja) atau *library* (perpustakaan kode) sampai dengan *game engine* (perangkat pengembangan *game*).

Dalam perkembangannya terdapat beberapa *framework* maupun *library* yang cukup populer dengan dukungan komunitas yang besar. Pada subbab ini akan dibahas beberapa diantara *framework* yang populer tersebut, sebagai referensi dasar bagi pembaca yang ingin mengembangkan kemampuannya lebih lanjut di bidang pengembangan game. Berikut *framework* dan *library* terkait pengembangan aplikasi/*game* dengan *javascript* dan HTML5 *canvas* :

1. GDevelop
2. melonJS
3. impactJS
4. PhaserJS
5. BabylonJS
6. PixyJS
7. PlayCanva
8. Construct
9. Aframe, dsb

Framework /Library di atas beberapa diantaranya memiliki fokus untuk menjadi alat pengembang (*authoring tool*) untuk berbagai jenis pengguna (dari pemula hingga mahir), *Framework /Library* menyediakan seperangkat alat yang lengkap untuk membantu dalam mengembangkan *game game* 2D, seperti *platformer*, *shoot'em up*, sampai *game* 3D (Ramtal dkk, 2014). Beberapa *framework* juga memungkinkan pengguna untuk mengekspor *game*/aplikasi ke berbagai *platform*, seperti Android, iOS, Facebook Instant *Game*s, dan lainnya.

Pada buku ini digunakan *library* dasar **gameLib.js** yang dikembangkan sendiri oleh penulis dengan tujuan khusus pengembang pemula. *Library* dibuat secara sederhana dengan menggunakan fungsi berbahasa Indonesia untuk meningkatkan pemahaman pembaca dan agar memahami konsep-konsep dasar dalam pengembangan aplikasi/game. Pada tahapan selanjutnya melalui *library* **gameLib.js** diharapkan pembaca dapat mengeksplorasi lebih dalam terkait arsitektur tingkat rendah dari pengembangan *game*/aplikasi.

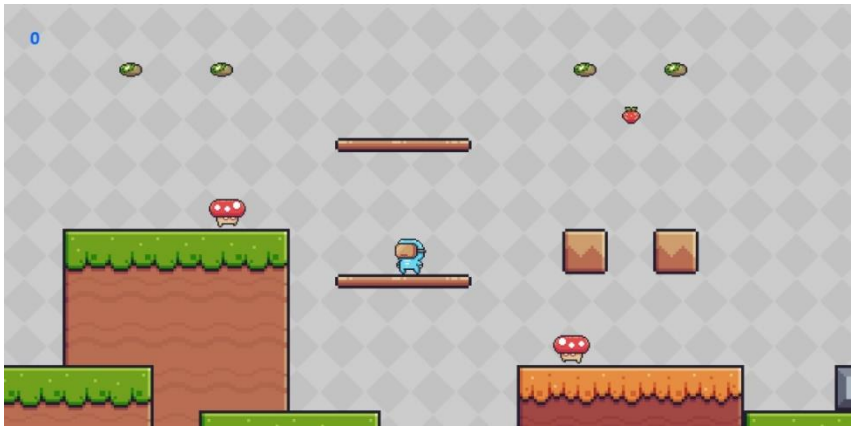
1.5 Cakupan Bahasan dalam Buku Ini

Buku ini secara spesifik menjelaskan tentang dasar pengembangan *game* berbasis HTML5 dengan bahasa pemrograman *Javascript*. Buku dibagi menjadi beberapa bab yang akan menjabarkan proses pengembangan *game* sebagai berikut :

1. Tutorial-1 membahas pembuatan *game* bertipe *Shoot'em up*, pemain mengendalikan pesawat dan diharuskan menghancurkan musuh sebanyak mungkin untuk meraih nilai tertinggi. Pengembangan game akan dijelaskan dalam beberapa bab meliputi gerakan dasar, objek kompleks, penambahan suara dan antar muka.

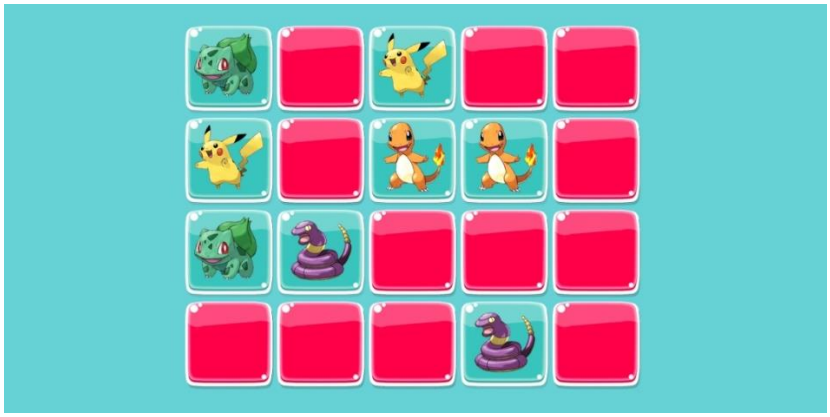
Gambar 3. Hasil tutorial-1

2. Tutorial-2 akan membahas pembuatan game bertipe *platformer*. Penjelasan proses dimulai dari gerakan dasar karakter, pengembangan level permainan dan pemrograman gerakan dalam area permainan berbasis *tile*.



Gambar 4. Hasil tutorial-2

3. Tutorial-3 akan membahas pembuatan game mencocokan gambar. Pengembangan game menggunakan teknik *flowchart* dimana fungsi-fungsi *game* telah dirancang secara mendetail di awal proyek.



Gambar 5. Hasil tutorial-3

4. Tutorial-4 akan membahas tentang pembuatan kuis interaktif. Pada tutorial ini dijelaskan teknik *porting* dari bahasa pemrograman lain (*Actionscript*) menuju ke *Javascript*. Proses mengalihbahasa dijelaskan secara sederhana untuk mempermudah proses adaptasi pemrograman.

Gambar 6. Hasil tutorial-4

5. Tutorial-5 menjelaskan proses menggerakan pemain pada permainan papan (*board game*). Pada tutorial 5 dijelaskan tentang prinsip menggerakkan karakter permainan melalui perhitungan matematis sederhana.

Gambar 7. Hasil tutorial-5

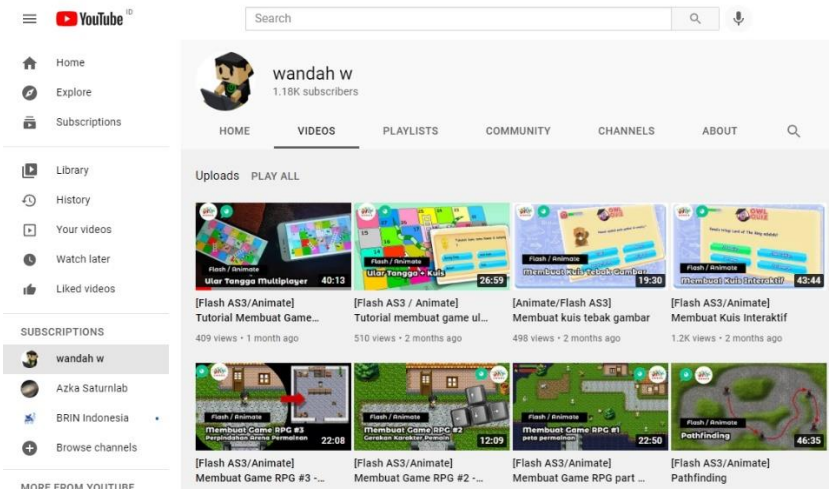
Pada tahapan selanjutnya buku ini dapat diintegrasikan dengan buku yang ditulis sebelumnya, yaitu "Game Edukasi RPG (*Role Playing Game*)" dan "Laboratorium Virtual – konsep dan pengembangan simulasi Fisika".

1.6 Target Pembaca

Pada awal penulisan, buku ini ditujukan untuk pengembang *game* pemula. Namun penulis menyadari sepenuhnya bahwa pada saat buku ini ditulis, para pembelajar *game* merupakan generasi baru (generasi milenial, *alpha* dan sesudahnya), yang relatif menginginkan sesuatu yang sifatnya instan, sehingga tutorial yang ada dalam buku ini diupayakan menggunakan kode seminimal mungkin melalui optimalisasi *framework*.

Pada saat buku ini ditulis, dapat ditemukan beberapa aplikasi pengembang *game* berbasis HTML 5 yang sangat mudah dioperasikan dan menggunakan metode *visual scripting* (memanipulasi kode menggunakan tampilan grafis) seperti Construct, GDevelop, GameMaker dan sejenisnya. Kendati demikian penulis beranggapan bahwa mempelajari kode dasar dapat menjadi pondasi yang kuat. Seseorang yang belajar kode dengan arsitektur rendah (pengkodean dasar), akan lebih mudah memahami logika yang ada di balik satu aktivitas tertentu. Oleh karena itu buku ini mencoba menjelaskan arsitektur rendah tersebut kepada pembaca disertai dengan penjelasan-penjelasan di tiap baris kode.

Pada saat buku ini ditulis, popularitas buku sebagai sumber belajar pemrograman juga semakin tergeser dengan video tutorial yang terdapat di berbagai *platform* digital. Oleh karena itu, selain melalui tulisan di dalam buku ini penulis juga menyajikan tutorial dalam bentuk video melalui kanal YouTube *channel* pada tautan
https://www.youtube.com/user/wandahw/videos



Gambar 8. *Channel* YouTube untuk menunjang pembelajaran

1.7 File Sumber Buku

File sumber buku untuk mengembangkan masing-masing tutorial terdiri dari 2 jenis, yaitu kode dan aset visual. Kode dapat di tulis ulang maupun di*copy-paste* ke dalam *file* baru untuk mempermudah proses pengembangan. Selain itu kode juga telah disediakan pada situs www.wandah.org/code

Sementara untuk aset visual terdapat beberapa hal yang perlu diperhatikan. Beberapa tutorial yang ada di dalam buku menggunakan gambar yang dibuat penulis secara mandiri dan sumber gambar yang didapatkan dari mengunduh aset yang tersedia secara *online*. Aset *online*, sebagai contoh aset yang digunakan dalam tutorial-2 diunduh dari situs itch.io dan memiliki lisensi "*permissive lisence*" (CC-BY) yang memperbolehkan untuk digunakan untuk proyek pengembangan *game*, namun asset tidak diperbolehkan untuk didistribusikan di luar situs penyedia. Sehingga pada buku ini aset visual harus dibuat secara mandiri oleh pembaca, atau pembaca dapat mengunduh sendiri aset yang dibutuhkan.

1.8 Lisensi Buku

Buku ini diterbitkan secara *online* dalam bentuk *e-book* dan diedarkan secara gratis, dengan beberapa ketentuan sebagai berikut :

1. Pembaca diperbolehkan menyebarkan dan memperbanyak buku tanpa mengubah, menambah atau mengurangi konten buku.
2. Pembaca diperbolehkan mencetak buku untuk keperluan pribadi, dan tidak diperbolehkan mencetak untuk diperjual belikan (dilarang meng-komersialkan buku ini dalam bentuk apapun).

   Pengambilan keuntungan sepihak melalui komersialisasi buku tanpa seijin penulis akan mengerdilkan ilmu, mengkhianati konsep berbagi ilmu dan mengurangi semangat penulis untuk berbagi lagi pada buku-buku selanjutnya.
3. Penambahan atribut nama penulis (*credit*) diperlukan pada setiap aplikasi yang dihasilkan.
4. Pembaca diperbolehkan menggunakan dan memodifikasi file tutorial untuk keperluan apapun, termasuk keperluan komersial.

HTML 5 Game Project

Game HTML 5 memiliki struktur proyek yang sederhana, terdiri dari file html, kode javascript serta seperangkat asset audio visual

BAB 2

PENGATURAN PROYEK PENGEMBANGAN *GAME*

2.1 Pengaturan Proyek (*Project Setup*)

Dalam membuat sebuah *game* atau aplikasi interaktif berbasis HTML5 kita akan membutuhkan beberapa jenis *file*, yaitu *file* HTML, *file Javascript*, *file* gambar dan *file* suara. Apabila anda pernah mengembangkan aplikasi menggunakan *authoring tools* seperti Adobe Flash/Adobe Animate, Unity, Construct dan sejenisnya maka akan dipahami bahwa terdapat beberapa *file* baik berupa *file* kode, maupun *file* asset (gambar dan suara) yang akan di *compile* menjadi 1 kesatuan proyek *file*. Hal ini sedikit berbeda apabila kita mengembangkannya dengan HTML dan JS tanpa *authoring tools*. Dalam aplikasi HTML *canvas*, *file* pada umumnya tetap terpisah dan berada di *folder -folder* tertentu. Sehingga kita membutuhkan managemen pengaturan *file* yang baik agar proses kerja dapat berlangsung efektif (Feil dan Scattergood, 2005).

Dalam buku ini akan digunakan struktur *folder* sederhana, dimana masing-masing proyek akan membutuhkan satu *folder* khusus, yang di dalamnya terdapat 1 buah *file* HTML dan 2 buah *folder* , yaitu *folder* **assets** dan *folder* **js**. *Folder* **assets** akan digunakan untuk menampung asset visual berupa gambar dan *file* audio. Sedangkan *folder* **js** secara khusus digunakan untuk menyimpan *file* kode *javascript* yang akan digunakan untuk mengendalikan permainan. Struktur *folder* dapat anda lihat pada gambar berikut :

Gambar 9. Struktur *folder*

2.2 *File* HTML

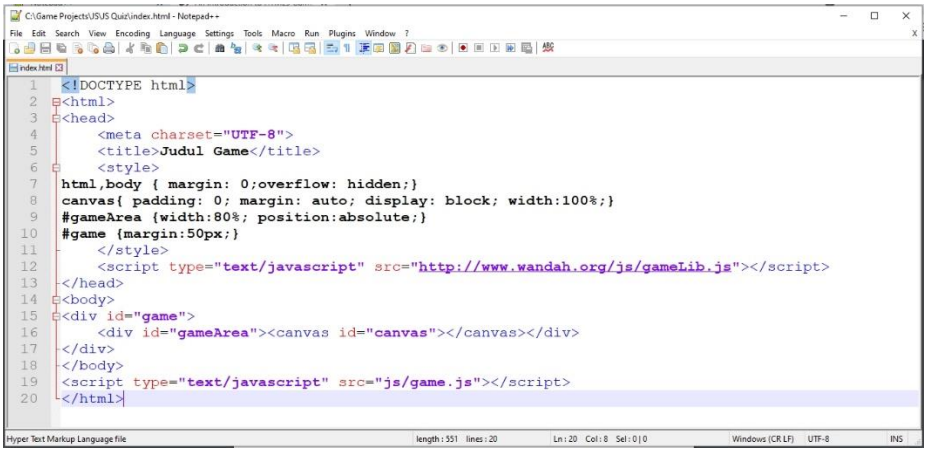
File HTML pada proyek yang dibuat, merupakan *file* yang akan dijalankan melalui *web browser*. Pada umumnya *file* ini diberi nama **index.html** namun demikian kita dapat

memberikan nama lain selama ekstensi *file* adalah bertipe HTML. Struktur *file* HTML yang akan digunakan pada tutorial di buku ini secara umum memiliki konten yang sama, yaitu :

```
1. <html>
2. <head>
3.      <meta charset="UTF-8">
4.      <title>Judul Aplikasi</title>
5.      <style>
6. html,body { margin: 0;overflow: hidden;}
7. canvas{ padding: 0; margin: auto; display: block; width:100%;}
8. #gameArea {width:80%; position:absolute;}
9. #game {margin:50px;}
10.     </style>
11.     <script type="text/javascript"
                    src="http://www.wandah.org/js/gameLib.js"></script>
12. </head>
13. <body>
14.      <div id="game">
15.          <div id="gameArea"><canvas id="canvas"></canvas></div>
16.       </div>
17. </body>
18. <script type="text/javascript" src="js/game.js"></script>
19. </html>
```

Pada baris 5-10 terdapat pengaturan penggayaan (*style*) atau CSS, yang secara umum juga dapat dilakukan secara terpisah dengan membuka *file* bertipe CSS. Anda juga dapat melakukan pengaturan ukuran area permainan melalui tag atau melakukan penempatan gameArea melalui tag .

Dalam beberapa sumber yang dapat ditemukan di beberapa situs tutorial, beberapa pengembang meletakkan baris *script* (seperti pada baris 18) di dalam tag `<body>` atau melakukan pemanggilan kode melalui *event* `window.onload`. Metode apapun yang dilakukan pada dasarnya benar, karena *Javascript* memiliki fleksibilitas sesuai dengan kecenderungan masing-masing pengembang.



Gambar 10. *File* **index.html** pada aplikasi Notepad++

2.3 *File Javascript* (.js)

Folder **js** digunakan untuk menyimpan *file* bertipe *javascript* (.js). Pada proyek *game*/aplikasi yang akan dikembangkan di dalam buku ini setidaknya terdapat 2 *file* js, yaitu *file* **gameLib.js** yang diunduh secara langsung dari situs wandah.org atau anda unduh terlebih dahulu kemudian meletakkannya ke *folder* **js**. *File* kedua adalah *file* untuk mengatur interaktivitas pada *canvas* (lihat baris 18 pada *file* HTML di atas).

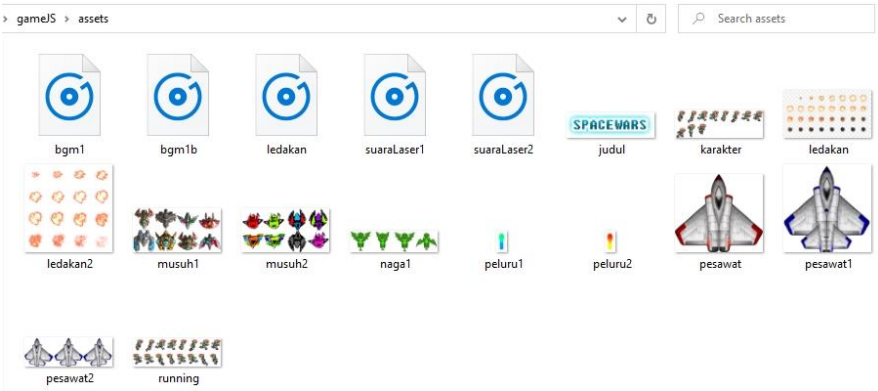
2.4 *File* Assets

Pada *folder* **assets** diletakkan seluruh aset audio visual yang akan digunakan di dalam aplikasi/permainan. *File* aset berupa gambar dapat dibuat dengan format JPG atau PNG apabila membutuhkan gambar dengan *background* transparan. Untuk membuat gambar berseri (berurutan, untuk membuat gambar animasi) akan lebih mudah apabila gambar ditampilkan dalam format *spritesheet* . *Sprite sheet* merupakan sebuah gambar yang terdiri dari beberapa gambar yang lebih kecil (*sprite*). Menggabungkan gambar kecil dalam satu gambar besar bertujuan untuk meningkatkan kinerja *game*, mengurangi penggunaan memori, dan mempercepat waktu mulai dan memuat game. Perhatikan contoh berikut :

Gambar 11. *Spritesheet*

Pada gambar 4 (kiri) terdapat satu set gambar karakter yang secara umum dapat digunakan dalam *game* RPG 2 dimensi. Satu set gambar karakter berjalan dengan 4 arah dan masing-masing memiliki 3 gerakan, digabung menjadi sebuah *file* PNG. Hal ini akan menjadikan *game*/aplikasi menjadi lebih optimal dalam penggunaan memori. Pada gambar 4 (kanan) terdapat satu set panel yang digunakan sebagai GUI (*Graphical User Interface*). Sebuah *spritesheet* dapat berisi gambar satu jenis (satu tema) atau berlainan, yang terpenting adalah ukuran gambar tiap-tiap *sprite* (gambar kecil) dapat didefinisikan dengan mudah.

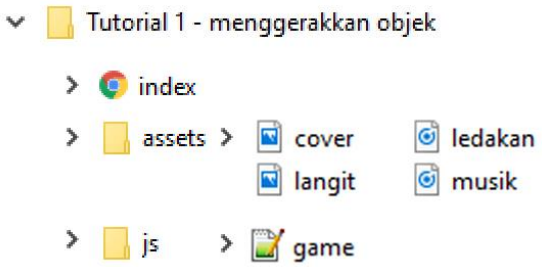
Untuk aset suara digunakan *file* suara bertipe WAV atau MP3 dan dapat diletakkan di dalam *folder* **assets**. Anda dapat juga memisah antara *file* suara dan *file* gambar, namun dalam tutorial di dalam buku ini, untuk mempermudah pemahaman *file* gambar dan suara diletakkan di dalam satu *folder* yang sama.



Gambar 12. *File* audio dan gambar dalam *folder* **assets**

2.5 Struktur *Folder* Proyek *Game* HTML 5

Setelah memahami masing-masing *file* penyusun proyek game HTML, maka *file-file* tersebut disusun ke dalam *folder*. Struktur *folder* proyek *game* yang akan dibuat adalah sebagai berikut:



Gambar 13. Struktur *folder* proyek *game* HTML 5

[halaman ini sengaja dikosongkan]

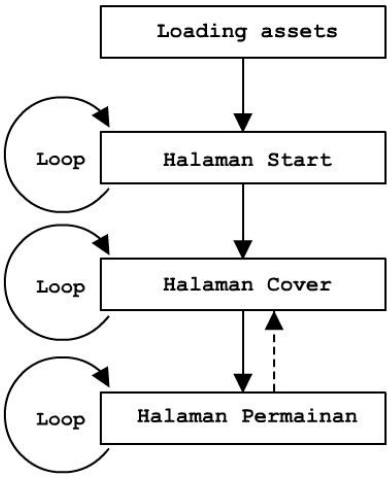
Skill Dasar

Belajar pembuatan game memiliki tingkatan dan metode yang berbeda-beda.
Skill dasar pemrograman mutlak diperlukan jika ingin menguasai pembuatan game dengan cepat

BAB 3
GERAKAN OBJEK

3.1 Memahami Proses Kerja *Canvas*

Proses kinerja dasar *javascript* dalam mengelola *canvas* adalah dengan menggambar *canvas* secara berulang-ulang. Namun sebelumnya pada tahapan awal diperlukan kode untuk mengunduh seluruh aset yang digunakan di dalam *game*/aplikasi. Setelah aset terunduh, maka aset dimanipulasi posisi, ukuran dan bentuk tampilannya, kemudian di*render* ke *canvas*. Secara sederhana proses kerja *canvas* adalah sebagai berikut :



Gambar 14. Proses kerja *canvas* dalam *game*/aplikasi

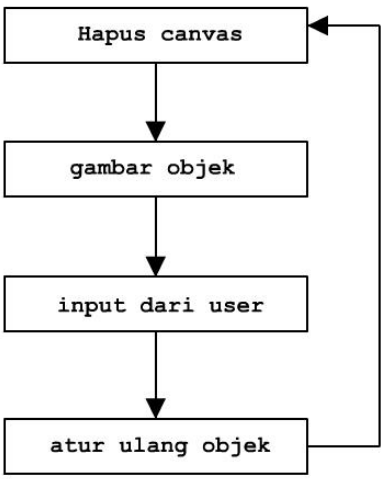
Pada tahapan awal diperlukan proses *preload* untuk mengunduh aset gambar maupun aset suara. Dalam mode *offline* proses ini tidak akan memakan banyak waktu, sehingga berlangsung dengan cepat, namun dalam mode *online* aset harus diunduh satu demi satu dan membutuhkan beberapa waktu. Oleh karena itu pada tahapan ini akan ditampilkan progres dari proses pengunduhan. Ketika proses pengunduhan selesai, maka aplikasi akan diarahkan ke halaman *start*.

Gambar 15. Proses *loading* aset

Halaman *start* perlu ditambahkan dalam aplikasi berbasis *canvas*. Pada halaman ini dapat ditampilkan logo dan minimal terdapat sebuah tombol "*start*". Hal ini diperlukan agar terjadi interaksi pertama antara pengguna dengan objek DOM HTML (*canvas*). Di sebagian besar *browser* terdapat peraturan terkait keamanan bahwa kode *js* tidak bisa mengeksekusi kode-kode tertentu sebelum berinteraksi dengan objek DOM. Sebagai contoh, *js* tidak boleh memainkan suara seketika setelah proses *loading* musik selesai, atau *js* tidak boleh menjadikan aplikasi menjadi *fullscreen* sebelum pengguna berinteraksi dengan objek DOM. Hal ini sangat logis, mengingat apabila diberikan ijin untuk mengakses kode-kode tertentu tanpa interaksi awal dari pengguna, *js* dapat disalahgunakan seperti pencurian data atau menyebarkan aplikasi berbahaya seperti virus/*malware*. Atas peraturan keamanan tersebut, pada *game* HTML5 sering kali diawali dengan tombol *start* untuk masuk ke halaman awal (judul) permainan.

Gambar 16. Proses kerja *game* HTML5 *canvas*

Pada halaman judul (*cover*) dan halaman permainan dilakukan proses berulang (*loop*) untuk menggambar ulang *canvas* (*rendering*) hingga didapatkan tampilan yang diinginkan. Proses berulang tersebut pada umumnya adalah sebagai berikut :



Gambar 17. Proses berulang rendering *canvas*

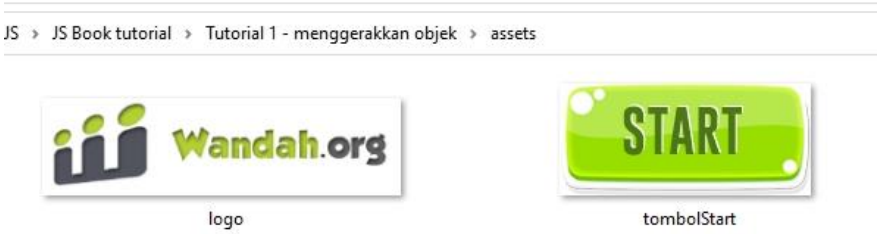
Pada dasarnya *canvas* adalah media yang digunakan oleh *javascript* untuk menampilkan gambar-gambar objek. *Canvas* harus dihapus dan digambar ulang setiap saat sesuai dengan objek-objek yang dipakai dan diatur oleh kode *js* (Takatalo, 2015).

3.2 Tutorial 1 - Membuat halaman start

Pada **tutorial-1** kita akan membuat sebuah gerakan objek sederhana, yaitu gerakan pesawat. Pesawat akan dapat dikendalikan oleh pemain menggunakan *keyboard* maupun menggunakan *virtual joystick*.

Tahapan pertama dalam memahami pembuatan *game* adalah memahami bagaimana cara menambahkan objek ke layar. Pada tutorial ini akan dibuat sebuah halaman *start* yang memiliki sebuah logo dan sebuah tombol. Perhatikan langkah-langkah berikut :

1. Siapkan sebuah *folder* berikut kelengkapan di dalamnya (1 *file* **index.html**, *folder* **assets** dan *folder* **js**), sebagaimana pengaturan *folder* proyek pada bab 2.5.
2. Siapkan 2 aset gambar berupa gambar **logo.png** dan gambar **tombolStart.png** (perhatikan gambar. Anda dapat membuat aset gambar tersebut dengan menggunakan aplikasi grafis seperti Photoshop, Adobe Ilustration, Paint atau aplikasi lainnya).



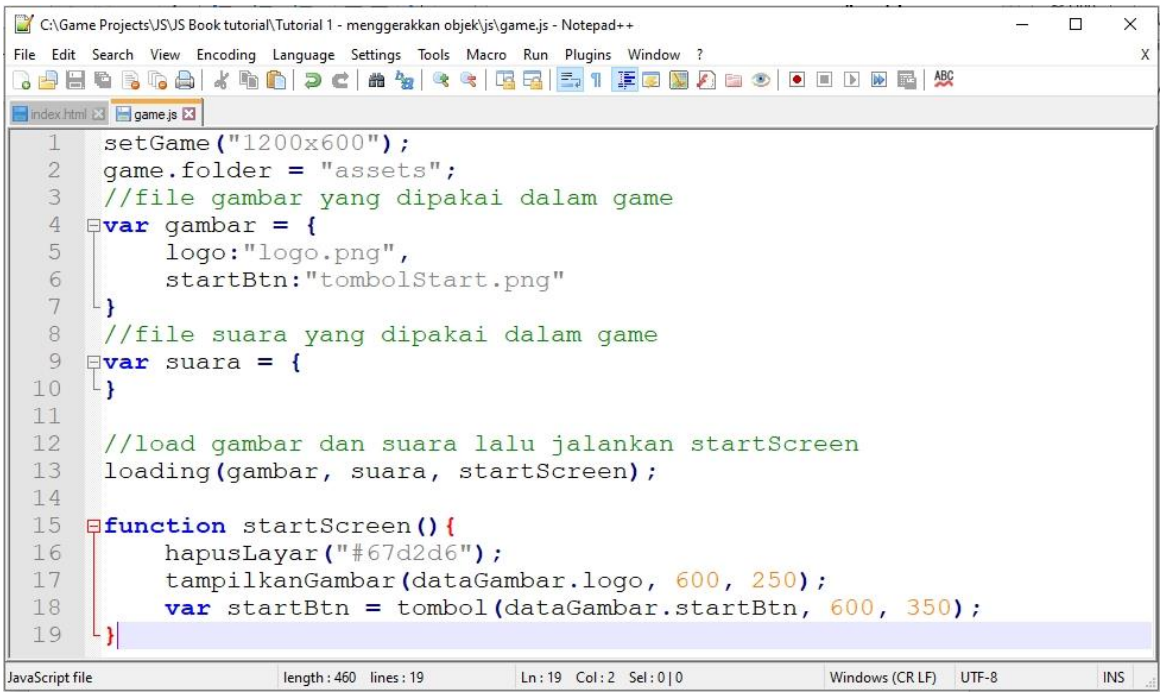
Gambar 18. Aset gambar

3. Letakkan kedua gambar tersebut pada *folder* **assets**.
4. Buka aplikasi Notepad++. Kemudian buat *file* baru dan ketikkan kode berikut :

```
1. setGame("1200x600");
2. game.folder  = "assets";
3. //file gambar yang dipakai dalam game
4. var gambar = {
5.        logo:"logo.png",
6.        startBtn:"tombolStart.png"
7. }
8. //file suara yang dipakai dalam game
9. var suara = {
10. }
11.
12. //load gambar dan suara lalu jalankan startScreen
```

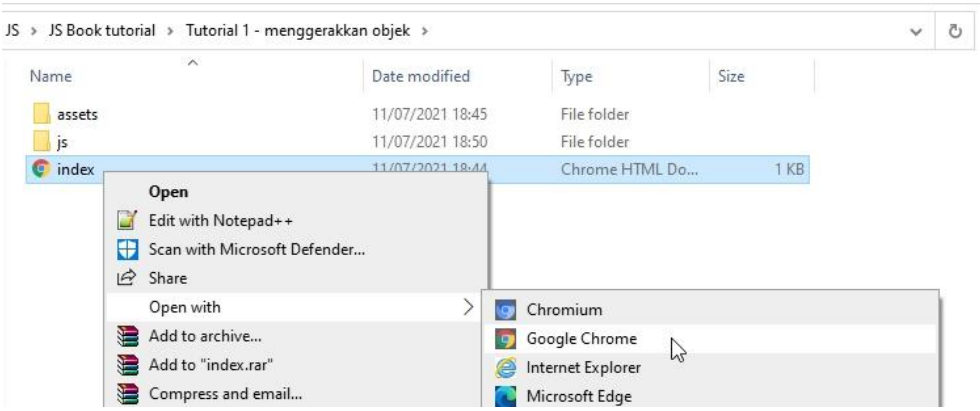
```
13. loading(gambar, suara, startScreen);
14.
15. function startScreen(){
16.         hapusLayar("#67d2d6");
17.         tampilkanGambar(dataGambar.logo, 600, 250);
18.         var startBtn = tombol(dataGambar.startBtn, 600, 350);
19. }
```

5. Simpan kode tersebut pada *folder* **js** dengan nama game.**js**



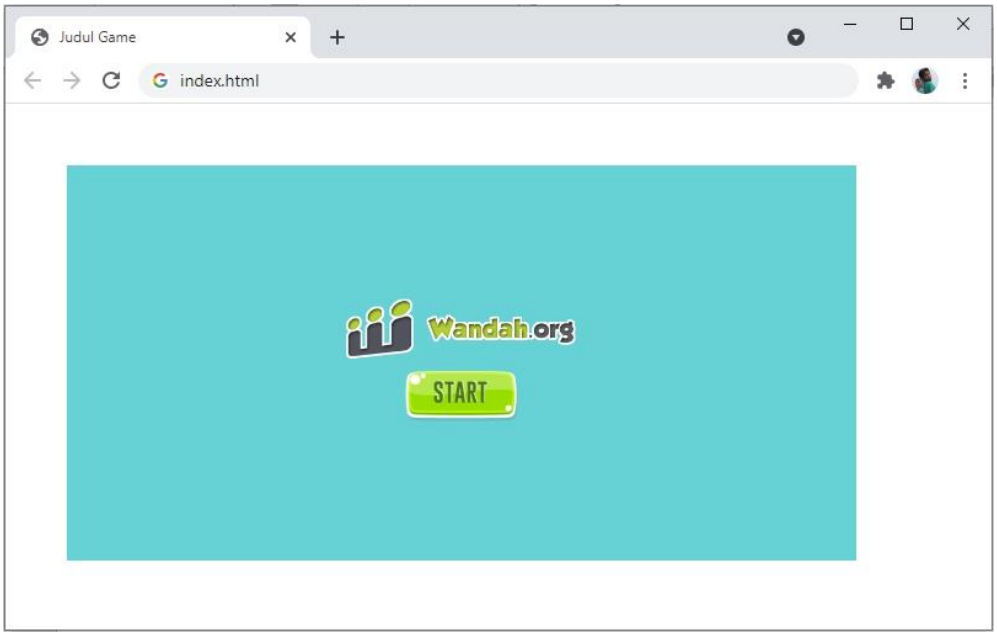
Gambar 19. Tampilan kode game.**js** pada Notepad++;

6. Buka *folder* tempat anda bekerja dengan Windows Explorer, kemudian jalankan *file* **index.html** dengan cara *double-click file* tersebut, dan *file* akan terbuka pada *web browser* aktif. (Cara lain adalah dengan klik kanan *file* **index.html** dan memilih menu **open with>...**).



Gambar 20. Menjalankan *file* **index.html**

7. Pada halaman *browser* akan ditampilkan hasil kompilasi dari kode game.**js** yaitu berupa gambar logo dan tombol yang berada di tengah layar. Pada tahapan ini tombol *start* belum bisa ditekan karena kita belum menambahkan kode untuk melakukannya.



Gambar 21. Tampilan *canvas* di *web browser*

Penjelasan program

```
1. setGame("1200x600");
2. game.folder  = "assets";
```

Pada baris 1 dilakukan pengaturan ukuran *game* yaitu 1200 x 600 *pixel*. Anda dapat melakukan perubahan pada angka tersebut dan menyesuaikan sesuai kebutuhan. Hal yang perlu diperhatikan adalah dalam kode **index.html** struktur tag `<canvas>` berada di dalam tag `<div id="gameArea">` yang diatur memiliki lebar 80% dari lebar layar *browser* (lihat baris 8: `#gameArea {width:80%; position:absolute;}` pada *file* **index.html**). Hal ini menyebabkan ukuran layar *canvas* tidak ditampilkan sebesar 1200x600 *pixel*, namun sesuai dengan pengaturan gameArea, yaitu 80% dari lebar layar. Anda dapat mengubah *style* dari gameArea tersebut sesuai kebutuhan dengan mengatur *width* dan *height* nya.

Maksud dari peletakan tag `<canvas>` ke dalam tag `<div id="gameArea">` adalah untuk menghasilkan tampilan yang responsif (mudah menyesuaikan dengan ukuran layar apapun). Selain itu pada tahapan *publishing* nantinya, pengaturan struktur *tag* HTML seperti ini akan mempermudah pengembang web (*web developer*) untuk meletakkan konten *game* yang kita buat.

Pada baris ke-2 dilakukan pengaturan *folder* **assets**, sehingga kode akan membuka aset melalui *folder* tersebut. Baris ini sebenarnya boleh tidak dituliskan, karena secara *default* variabel `game.folder` bernilai "`assets`". Namun apabila pengguna ingin menggunakan nama *folder* lain, maka baris ini harus diketikan sesuai dengan nama *folder* yang dimaksud.

```
4. var gambar = {
5.         logo:"logo.png",
6.         startBtn:"tombolStart.png"
7. }
8. //file suara yang dipakai dalam game
9. var suara = { }
```

Pada baris 4 dan 9 dilakukan pengaturan *variabel* gambar dan suara. *Variabel* gambar berisi identifikasi dari masing-masing gambar yang ditambahkan ke dalam game. Sebagai contoh *file* gambar "**logo.png**" diidentifikasi sebagai `logo`, dan *file* "**tombolStart.png**" diidentifikasi sebagai `startBtn`. Untuk menggunakan *file* gambar tersebut nantinya digunakan kode `dataGambar.logo` atau `dataGambar.startBtn`. Sementara pada baris 9, *variabel* suara masih kosong, karena kita belum menambahkan *file* suara.

```
13. loading(gambar, suara, startScreen);
```

Pada baris 13 digunakan kode *loading* untuk membuka aset gambar dan suara, dan ketika proses *loading* selesai maka akan dijalankan fungsi `startScreen`.

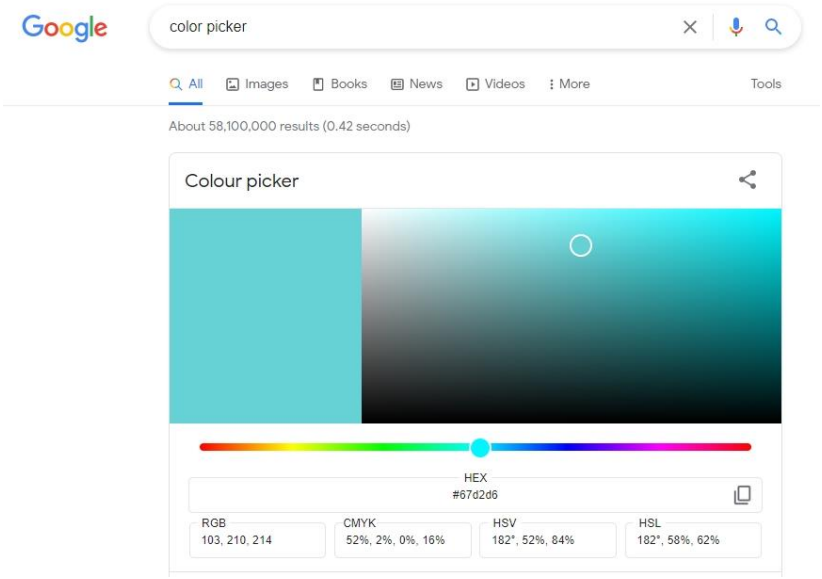
```
15. function startScreen(){
16.         hapusLayar("#67d2d6");
17.         tampilkanGambar(dataGambar.logo, 600, 250);
18.         var startBtn = tombol(dataGambar.startBtn, 600, 350);
19. }
```

Baris 15 disebut sebagai fungsi. Fungsi merupakan kumpulan baris kode (kode yang terletak di antara tanda { … }) yang akan dijalankan dalam satu waktu. Pada fungsi `startScreen` di atas di dalamnya terdapat 3 baris kode, yaitu :

1. `hapusLayar(warna);`

   digunakan untuk menghapus keseluruhan *canvas* dan mengisinya dengan warna (dalam hal ini menggunakan kode warna heksa). Untuk mendapatkan kode warna tersebut, anda dapat membuka Google dan mengetikkan *keyword* "*color picker*". Anda dapat menyalin kode warna yang ada pada panel HEX dan meletakkannya di antara

dua tanda kurung (“`kode HEX warna`”). Apabila kode warna tidak diisikan, maka akan ditampilkan warna hitam.



Gambar 22. *Color picker* pada halaman web

2. `tampilkanGambar(dataGambar.nama, posisiX, posisiY);`

digunakan untuk menampilkan gambar di posisi x dan y (kordinat layar). Pada baris 17 tersebut kode x bernilai 600 dan kode y bernilai 250. Perhatikan sistem kordinat layar berikut :

Gambar 23. Sistem kordinat layar

Sistem kordinat layar *canvas* dimulai dari titik pojok kiri atas (koordinat 0,0). Sumbu x positif ke arah kanan (warna merah), dan sumbu y positif ke arah bawah (warna biru) . Ketika sebuah gambar diletakkan pada sumbu x = 600 dan y = 250, maka

gambar akan diletakkan secara horisontal 600 *pixel* ke kanan, dan secara vertikal 250 *pixel* ke bawah, dengan pusat kordinat tepat di titik tengah gambar.

3. `var startBtn = tombol(dataGambar.startBtn, 600, 350);`
   digunakan untuk membuat tombol bernama `startBtn` yang menggunakan `dataGambar.startBtn` sebagai tampilan tombol, dan diletakkan pada koordinat x = 600 dan y = 350.

3.3 Membuat halaman judul (*cover/tittle*)

Tahapan selanjutnya adalah membuat halaman judul permainan (*cover/tiitle*). Pada halaman ini akan ditampilkan gambar latar dan judul permainan serta satu buah tombol untuk memulai permainan. Sebelum beranjak pada proses pemrograman, terlebih dahulu disiapkan aset visual (gambar) yang akan digunakan untuk halaman ini.

Gambar 24. Gambar **cover.jpg** dan **btn-play.png**

Setelah aset visual disiapkan, lanjutkan proyek **tutorial-1** sebagai berikut :

1. Letakkan *file* **cover.jpg** dan **btn-play.png** ke dalam *folder* **assets**.

Gambar 25. Isi *folder* **assets** saat ini.

2. Buka kembali *file* game.**js** dan tambahkan kode berikut (hanya yang di dalam kotak) :

```
1. setGame("1200x600");
2. game.folder  = "assets";
3. //file gambar yang dipakai dalam game
4. var gambar = {
5.         logo:"logo.png",
```

```
6.          startBtn:"tombolStart.png",
7.          cover:"cover.jpg",
8.          playBtn:"btn-play.png"
9.  }
10. //file suara yang dipakai dalam game
11. var suara = {
12. }
13.
14. //load gambar dan suara lalu jalankan startScreen
15. loading(gambar, suara, startScreen);
16.
17. function startScreen(){
18.         hapusLayar("#67d2d6");
19.         tampilkanGambar(dataGambar.logo, 600, 250);
20.         var startBtn = tombol(dataGambar.startBtn, 600, 350);
21.         if (tekan(startBtn)){
22.                 jalankan(halamanCover);
23.         }
24. }
25. function halamanCover(){
26.         hapusLayar("#67d2d6");
27.         gambarFull(dataGambar.cover);
28.         var playBtn = tombol(dataGambar.playBtn, 1100, 500);
29. }
```

Note : Tambahkan kode yang khusus diberikan kotak. Khusus untuk baris 6 terdapat karakter koma (,) di akhir kode.

3. Simpan *file* game.**js** tersebut, kemudian jalankan kembali *file* **index.html** di *web browser.* Pada tahapan ini halaman judul (*cover/tittle*) akan muncul setelah tombol *start* pada halaman *start* ditekan

Gambar 26. Halaman judul setelah menekan tombol *start*.

Penjelasan Program

```
6.        startBtn:"tombolStart.png",
7.        cover:"cover.jpg",
8.        playBtn:"btn-play.png"
```

Untuk menambahkan aset visual berupa gambar, diperlukan penambahan nama *file* ke dalam variabel gambar. Oleh karena itu pada akhir baris 6 ditambahkan karakter koma (,) dan dilanjutkan dengan nama *file* yang akan ditambahkan ke dalam aplikasi/game.

```
21.       if (tekan(startBtn)){
22.             jalankan(halamanCover);
23.       }
```

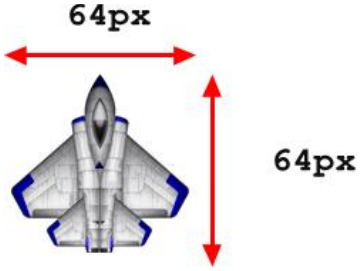
Pada tahapan sebelumnya (lihat baris 20) telah dibuat sebuah *variabel* `startBtn`. Untuk mengaktivasi tombol tersebut digunakan kode `tekan(nama tombol)` seperti pada baris 21. Setelah tombol ditekan oleh pengguna, maka akan dijalankan fungsi `halamanCover`.

```
25. function halamanCover(){
26.       hapusLayar("#67d2d6");
27.       gambarFull(dataGambar.cover);
28.       var playBtn = tombol(dataGambar.playBtn, 1100, 500);
29. }
```

Seperti halnya fungsi `startScreen`, pada fungsi `halamanCover` terdapat 3 baris kode. Yang membedakan adalah baris 27 yang menampilkan gambar secara penuh di *canvas* menggunakan fungsi `gambarFull`.

3.4 Menambahkan halaman permainan

Tahapan berikutnya adalah inti dari tutorial-1, yaitu membuat gerakan objek. Objek yang akan kita gerakkan adalah sebuah pesawat, oleh karena itu akan kita siapkan terlebih dahulu sebuah gambar pesawat dengan ukuran 64x64 *pixel*. Simpan dalam format PNG agar mendukung latar belakang transparan.



Gambar 27. Gambar **pesawat.png** sebagai objek yang akan digerakkan

Setelah aset visual disiapkan, lanjutkan proyek **tutorial-1** sebagai berikut :

1. Letakkan *file* **pesawat.png** ke dalam *folder* **assets**.
2. Buka kembali *file* game.**js** dan tambahkan kode berikut (dalam kotak) :

```
1.  setGame("1200x600");
2.  game.folder  = "assets";
3.  //file gambar yang dipakai dalam game
4.  var gambar = {
5.       logo:"logo.png",
6.       startBtn:"tombolStart.png",
7.       cover:"cover.jpg",
8.       playBtn:"btn-play.png",
9.       pesawat:"pesawat.png"
10. }
11. ...

26. function halamanCover(){
    27.  hapusLayar("#67d2d6");
    28.  gambarFull(dataGambar.cover);
    29.  var playBtn = tombol(dataGambar.playBtn, 1100, 500);
    30.  if (tekan(playBtn)){
    31.           setAwal();
    32.           jalankan(gameLoop);
    33.  }
    34.}
    35.
    36.function setAwal(){
    37.  game.jet = setSprite(dataGambar.pesawat);
    38.  game.jet.x = 600;
    39.  game.jet.y = 400;
    40.  game.kecJet = 5;
    41.}
    42.
    43.function gameLoop(){
    44.       hapusLayar();
    45.       sprite(game.jet);
    46.       if (game.kanan){
    47.              game.jet.x+=game.kecJet;
    48.       }else if (game.kiri){
    49.              game.jet.x-=game.kecJet;
    50.       }
```

```
51.         if (game.atas){
52.                 game.jet.y-=game.kecJet;
53.         }else if (game.bawah){
54.                 game.jet.y+=game.kecJet;
55.         }
56. }
```

3. Simpan *file* game.**js** tersebut, kemudian jalankan kembali *file* **index.html** di *web browser.* Pada tahapan ini pesawat akan dapat dikendalikan dengan menggunakan tombol panah *keyboard*.

Gambar 28. Gerakan pesawat menggunakan *keyboard*.

Penjelasan Program

```
30.                 if (tekan(playBtn)){
31.                         setAwal();
32.                         jalankan(gameLoop);
33.                 }
```

Pada baris 30, yaitu ketika tombol `playBtn` ditekan akan dijalankan 2 fungsi. Fungsi `setAwal` akan dijalankan satu kali, sedangkan fungsi `gameLoop` akan dijalankan secara berulang, karena kode `jalankan` akan membuat sebuah fungsi dijalankan secara terus menerus (*looping*).

```
36.       function setAwal(){
37.                 game.jet = setSprite(dataGambar.pesawat);
38.                 game.jet.x = 600;
39.                 game.jet.y = 400;
40.                 game.kecJet = 10;
41.       }
```

Pada fungsi `setAwal` dilakukan pengaturan objek dengan nama `game.jet`. Perpustakaan kode (*library*) **gameLib.js** memiliki sebuah objek dengan nama variabel *game* yang memiliki beberapa properti. Untuk mempermudah pengembang pemula, objek yang akan digerakkan di dalam permainan dapat dimasukkan ke dalam objek *game*, sehingga pengaturannya akan lebih fleksibel dan lebih mudah. Sebagai contoh pada baris 37 diatur sebuah objek baru di dalam objek *game*, yaitu `jet`. Objek `game.jet` diatur dengan fungsi `setSprite` memiliki gambar pesawat memiliki sumbu x (baris 38), dan sumbu y (baris 39). Selanjutnya juga di atur sebuah variabel `game.kecJet` (baris 40) untuk mengatur kecepatan gerak pesawat nantinya.

```
43.    function gameLoop(){
44.           hapusLayar();
45.           sprite(game.jet);
46.           if (game.kanan){
47.                  game.jet.x+=game.kecJet;
48.           }else if (game.kiri){
49.                  game.jet.x-=game.kecJet;
50.           }
51.           if (game.atas){
52.                  game.jet.y-=game.kecJet;
53.           }else if (game.bawah){
54.                  game.jet.y+=game.kecJet;
55.           }
56.    }
```

Akibat eksekusi kode `jalankan(gameLoop)` pada baris 32, maka fungsi `gameLoop` akan dijalankan secara terus menerus. Sesuai dengan penjelasan sebelumnya bahwa prinsip kerja *canvas* adalah dengan menghapus layar (baris 44), menggambar objek (baris 45), membaca input pemain (baris 46, 48, 51 dan 53), dan mengatur ulang posisi objek (baris 47, 49, 52 dan 54).

```
sprite(gameObject);
```

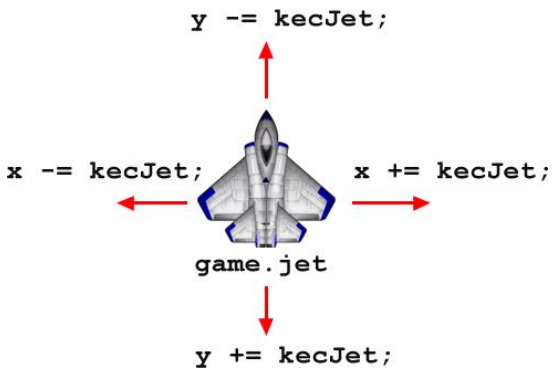
Kode `sprite` digunakan untuk menampilkan gambar pada *variabel* objek, yang dalam hal ini adalah gambar pesawat (pengaturan di baris 37). Pada bab selanjutnya, akan dijelaskan penggunaan kode `sprite` yang lebih kompleks, yaitu menampilkan gambar yang beranimasi.

Untuk membaca input pengguna dapat dilakukan dengan 3 cara, yaitu dengan *keyboard*, dengan *mouse* dan dengan sentuhan layar (*touch*). Ketiga cara tersebut sudah terdapat pada

library **gameLib.js** dan untuk membaca masing-masing masukan dapat dilakukan dengan menambahkan kode `game.kanan, game.kiri, game.atas, game.bawah` (untuk tombol panah atau *joystick virtual* yang akan dijelaskan pada bab selanjutnya) atau `game.keyCode` untuk membaca kode tombol yang ditekan.

```
if (game.kanan)
```

Logika `if` di atas, menyatakan bahwa apabila tombol *keyboard* panah kanan ditekan maka akan dilakukan kode pada blok `if ( ) { }` yang pada baris 47 terdapat `game.jet.x+=game.kecJet;` yaitu posisi `game.jet.x` digeser ke kanan (positif) sebesar variabel `game.kecJet` (5 *pixel*). Begitu juga halnya dengan logika `if` pada baris berikutnya yang akan menginisiasi perubahan posisi objek `game.jet`. Setelah posisi koordinat objek berubah, maka fungsi `gameLoop` akan diulang, *`canvas`* akan dihapus dan pesawat digambar pada posisi baru. Proses ini berlangsung dengan cepat (kecepatan *rendering* *`canvas`* diatur 30 *frame* perdetik, atau dapat diset dengan kode `game.fps = 30;`) sehingga pemain akan melihat gambar pesawat bergeser.



Gambar 29. Perubahan posisi objek pesawat

3.5 Menambahkan latar belakang (*background*).

Mengembangkan sebuah *game* bagi pemula akan lebih efektif apabila dipelajari secara bertahap dari hal yang bersifat fundamental (dasar). Setelah memahami gerakan pesawat di atas, selanjutnya dapat ditambahkan sebuah latar belakang (*background*) agar permainan terlihat menjadi lebih menarik. Untuk latar permainan diperlukan sebuah gambar langit dengan bintang-bintang, yang nantinya akan digerakkan. Untuk gambar latar pada *game* ini dibuat tidak terlalu besar (350x350 *pixel*), dan akan ditampilkan secara berulang (*tiling*). Perhatikan gambar **langit.jpg** yang digunakan sebagai latar.

Gambar 30. Gambar langit yang akan digunakan sebagai latar

Seperti halnya pada langkah sebelumnya, setiap kali aset baru ditambahkan, aset tersebut harus diletakkan dalam *folder* **assets**, dan selanjutnya dilakukan penambahan kode untuk mengidentifikasi *file* gambar.

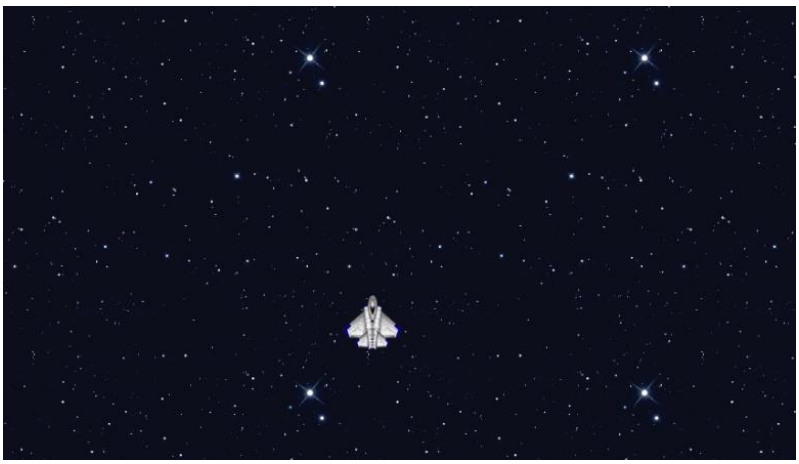
```
1. setGame("1200x600");
2. game.folder  = "assets";
3. //file gambar yang dipakai dalam game
4. var gambar = {
5.         logo:"logo.png",
6.         startBtn:"tombolStart.png",
7.         cover:"cover.jpg",
8.         playBtn:"btn-play.png",
9.         pesawat:"pesawat.png",
10.        langit:"langit.jpg"
11. }
```

Selanjutnya pada fungsi `gameLoop` perlu ditambahkan kode untuk menampilkan latar tersebut :

```
45. function gameLoop(){
46.        hapusLayar();
47.        latar(dataGambar.langit, 0, 5);
48.        sprite(game.jet);
49.        ...

59. }
```

Kode `latar(gambar, kecepatan x, kecepatan y)`, akan menampilkan gambar secara penuh di layar dan menggerakkannya sesuai pengaturan kecepatan. Dengan menggerakkan langit pada sumbu y sebanyak 5 *pixel*, maka latar belakang (*background*) langit akan bergerak ke bawah yang menimbulkan efek gerak pesawat ke depan.

Dalam penambahan kode latar tersebut, perlu diketahui urutan peletakan kode. Setelah *canvas* dihapus, latar belakang digambar terlebih dahulu, baru kemudian gambar pesawat. Kode yang ditulis lebih dahulu (di atas) akan dieksekusi terlebih dahulu, sehingga harus diperhatikan kapan dan dimana kode tersebut dituliskan.



Gambar 31. Hasil penambahan latar belakang (*background*)

3.6 Membatasi gerakan pesawat.

Pada saat ini apabila tombol kanan pada *keyboard* ditekan secara terus menerus, maka pesawat akan bergerak ke kanan sampai keluar dari layar. Oleh karena itu perlu ditambahkan logika tertentu untuk membatasi gerakan pesawat agar tidak keluar dari layar. Untuk melakukannya perlu ditambahkan logika pada fungsi `gameLoop` sebagai berikut :

```
45. function gameLoop(){
46.         hapusLayar();
47.         latar(dataGambar.langit, 0, 5);
48.         sprite(game.jet);
49.         if (game.kanan && game.jet.x<game.lebar-game.jet.lebar/2){
50.                 game.jet.x+=game.kecJet;
51.         }else if (game.kiri && game.jet.x>game.jet.lebar/2){
52.                 game.jet.x-=game.kecJet;
53.         }
54.         if (game.atas && game.jet.y>game.jet.tinggi/2){
55.                 game.jet.y-=game.kecJet;
56.         }else if (game.bawah && game.jet.y<game.tinggi-game.jet.tinggi/2){
57.                 game.jet.y+=game.kecJet;
58.         }
59. }
```

Pada baris 49 ditambahkan logika `&&` (dan jika) nilai kordinat dari x masih kurang dari lebar *canvas* (`game.lebar`) dikurangi dari setengah lebar pesawat (`game.jet.lebar/2`).

Pada perhitungan tersebut digunakan setengah lebar pesawat, hal ini dikarenakan titik pusat perhitungan kordinat pesawat adalah tepat di tengah pesawat, sehingga diperlukan setengah lebar pesawat agar pesawat tepat utuh di dalam layar `canvas`. Pada baris 51, 54 dan 56 diberikan logika yang sama untuk mengatur batas gerak pesawat.

Pada titik ini, pemahaman tentang koordinat objek dan cara menggerakkan objek telah disampaikan. Hal ini menjadi salah satu dasar dalam pengembangan sebuah *game*/permainan.

Ekspektasi vs Skill

Tidak jarang pengembang game pemula ingin membuat game yang fantastis, kompleks, MMO, AAA. Seharusnya pengembang pemula memahami batasan skill yang dimiliki

BAB 4

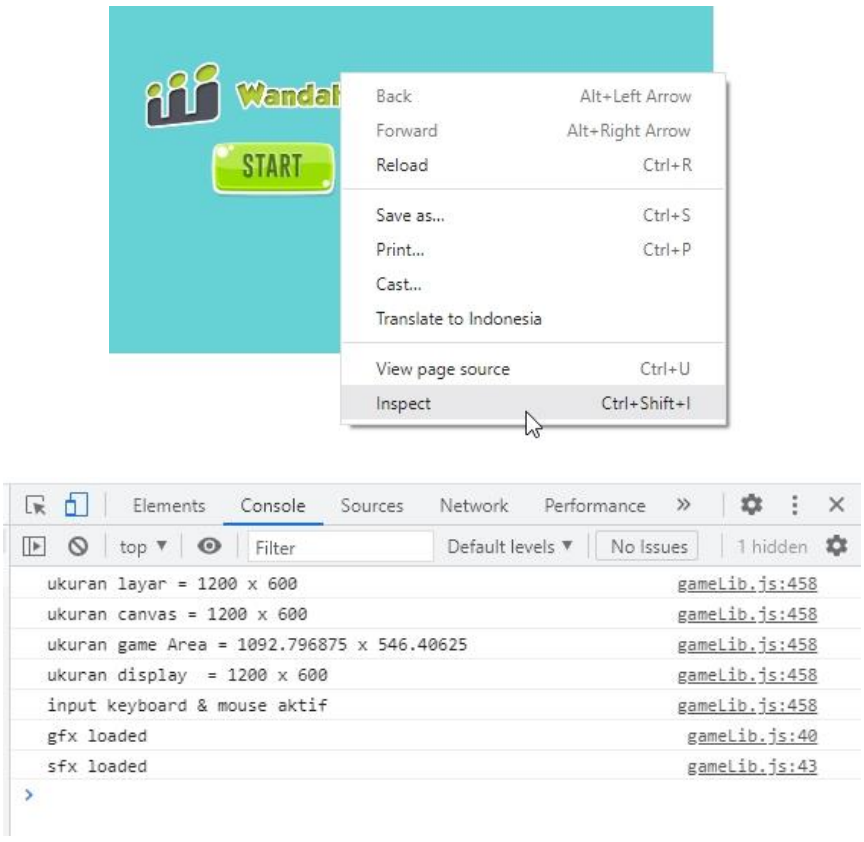
DEBUGGING

4.1 Memahami teknik *debugging*

Dalam pengembangan *game*/aplikasi dimungkinkan terdapat kesalahan penulisan kode yang mengakibatkan aplikasi/*game* tidak berjalan dengan baik atau tidak berjalan sama sekali. Proses menguji kesalahan pemrograman disebut sebagai *debugging* (Pedersen, 2003). Dalam proses *debugging* aplikasi berbasis HTML 5 dan *javascript*, dapat digunakan panel *console* pada *web browser*. Untuk membuka menu *console* dapat dilakukan dengan menekan *shortcut* sebagai berikut :

Mozilla Fire Fox	Ctrl+Shift+J
Google Chrome	Ctrl+Shift+J / F12
Internet Explorer	F12

Atau klik kanan pada *web browser* dan memilih menu *inspect* dan mengaktifkan panel *console*.

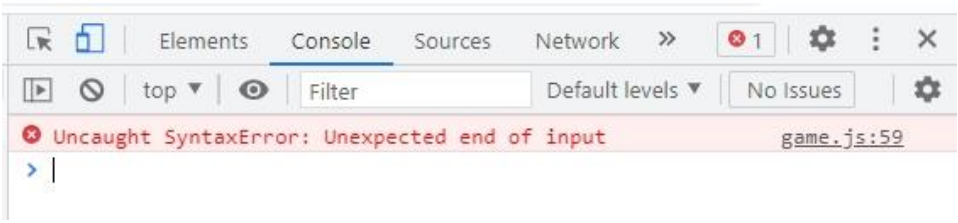


Gambar 32. Membuka *inspect* (atas) dan panel *console* (bawah)

Panel *console* tersebut sangat berguna untuk mengetahui kesalahan maupun untuk mengetahui proses yang telah dieksekusi oleh kode. Pada **tutorial-1** yang dilakukan pada bab 3 misalnya, apabila dibuka panel *console* maka akan muncul beberapa teks yang disebut sebagai *console log*. Dari *console log* ini kita dapat mengetahui proses yang telah terjadi atau yang dieksekusi dengan kode `console.log(teks yang akan dimunculkan);`

4.2 Menguji kesalahan

Apabila terdapat kesalahan dalam pengetikan kode, maka kode tidak dapat dieksekusi dan *browser* akan menampilkan layar kosong atau menampilkan sesuatu yang tidak sesuai dengan ekspektasi. Sebagai contoh buka kembali *file* game.**js** pada tutorial-1, kemudian hapus salah satu tanda `}`. Setelah itu simpan kembali dan buka *file* **index.html** melalui *web browser*. Maka *canvas* tidak akan ditampilkan dan pada panel *console* akan ditemukan kesalahan berupa teks berwarna merah.



Gambar 33. Pesan kesalahan pada panel *console*.

Apabila menemui kesalahan *syntax* atau kesalahan penulisan seperti contoh di atas, maka perlu mengacu pada posisi kesalahan tersebut, yang dalam contoh tersebut kesalahan terjadi pada *file* **game.js** baris ke 59 (`game.js:59`). Dengan cara ini kita dapat segera memperbaiki kesalahan yang ada, menyimpan ulang *file* **game.js** dan menjalankan kembali *file* **index.html** (*merefresh web browser*).

4.3 *Debugging* dengan teknik *print tracing*

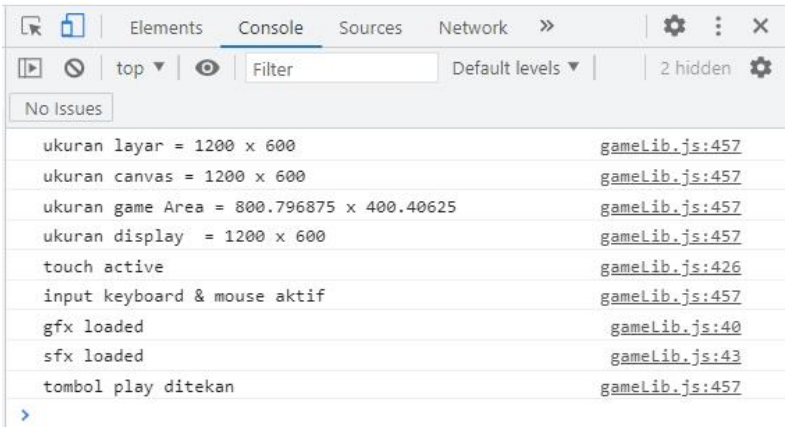
Salah satu metode *debugging* yang sering dipakai adalah teknik *print tracing*. Metode ini dilakukan dengan mengamati hasil *trace* atau hasil dari *console.log* saat kode dijalankan. Pada beberapa kasus, tidak terjadi kesalahan penulisan kode (kesalahan *syntax*), namun kode tidak berjalan sesuai ekspektasi. Dengan metode *tracing* tersebut, kita dapat mengetahui diposisi mana saat ini kode dijalankan, sehingga kita dapat memantau proses mana yang mengalami kesalahan. Dalam *library* **gameLib.js** untuk mempermudah proses *tracing* digunakan kode

```
trace(String);
```

Untuk lebih memahami penggunaannya, buka *file* **game.js** pada tutorial-1, kemudian sisipkan kode pada fungsi `halamanCover`. (perhatikan pada kode di dalam kotak)

```
27. function halamanCover(){
28.         hapusLayar("#67d2d6");
29.         gambarFull(dataGambar.cover);
30.         var playBtn = tombol(dataGambar.playBtn, 1100, 500);
31.         if (tekan(playBtn)){
32.                 trace("tombol play ditekan");
33.                 setAwal();
34.                 jalankan(gameLoop);
35.         }
36. }
```

Simpan kembali *file* **game.js** dan jalankan kembali **index.html** pada *web browser* dengan panel *console* terbuka. Pada saat tombol `playBtn` ditekan, akan muncul pesan "`tombol play ditekan`" pada panel *console*.



Gambar 34. Pesan *trace* pada panel *console*.

Tracing sangat bermanfaat dalam proses *debugging*, namun terkadang juga menghasilkan luaran yang terlalu banyak atau diistilahkan dengan nama *noisy*. Untuk mematikan fitur *trace* setelah proses *debugging* selesai, kita dapat menghapus kode masing-masing kode `trace`, atau dapat menggunakan kode :

```
game.debug = false;
```

yang diletakkan di awal kode setelah pengaturan `setGame`. Dengan kode tersebut, proses `trace` tidak akan menghasilkan luaran pada panel *console*.

Game Object

Setiap objek dalam game memiliki properties yang beragam seperti kordinat x, y, gambar Sprite, frame timer dan sebagainya

BAB 5
OBJEK KOMPLEKS

5.1 Menambahkan Peluru

Pada **tutorial-1** pesawat sudah dapat dikendalikan oleh pemain, namun pesawat belum memiliki fitur lain seperti menembakkan peluru. Pada tahapan ini fitur menembakkan peluru akan ditambahkan. Langkah pertama adalah menyiapkan aset visual untuk peluru, pada contoh ini digunakan 2 gambar yaitu **peluru1.png** (untuk peluru yang ditembakkan pemain) dan **peluru2.png** (untuk peluru yang ditembakkan musuh nantinya).

Gambar 35. Grafis untuk peluru, peluru1 (kiri) dan peluru2 (kanan)

Seperti pada langkah sebelumnya, *file* gambar harus diletakkan di dalam *folder* **assets**, dan selanjutnya dilakukan identifikasi aset agar dapat dikenali oleh kode. Buka *file* **game.js** dan tambahkan kode (kode dalam kotak).

```
4.  var gambar = {
5.          . . .
```

```
11.         peluru1:"peluru1.png",
12.         peluru2:"peluru2.png"
```

```
13. }
```

Dalam sebuah permainan/*game* bertipe *shoot'em up*, pemain maupun musuh dapat menembakkan peluru, sehingga pada saat yang bersamaan jumlah peluru di layar bisa lebih dari 1. Oleh karena itu, untuk mempermudah proses perhitungan posisi masing-masing peluru, diperlukan sebuah variabel yang meng*handle* seluruh peluru. Dalam praktik pembuatan *game*, data peluru tersebut dapat dimasukkan (di *push*) ke dalam *variabel* bertipe *Array*. Pengaturan *variabel* tersebut dapat diletakkan pada saat pengaturan awal di fungsi `setAwal`.

```
40. function setAwal(){
41.         . . .

45.         game.dataPeluru = [];
46. }
```

Pada kode baris 45 untuk membuat *variabel* baru bertipe *Array* cukup dengan menyebutkan nama *variabel* dan diikuti dengan tanda kurung siku (`[ ]`).

Untuk menembakkan peluru diperlukan input pemain, yang dalam hal ini menggunakan tombol spasi. Ketika tombol spasi ditekan, maka fungsi `tembak` akan dijalankan. Logika input untuk menembak tersebut perlu diletakkan pada fungsi `gameLoop`.

```
48. function gameLoop(){
49.         . . .

62.         //menembak
63.         if (game.spasi) tembak(game.jet, -5);
64.         aturPeluru();
65. }
```

Apabila anda perhatikan pada fungsi `gameLoop` di atas, terdapat 2 fungsi tambahan yang harus dibuat, yaitu fungsi `tembak` dan fungsi `aturPeluru`. Fungsi `tembak` digunakan untuk menginiasi awal peluru yang ditembakkan ketika tombol spasi ditekan. Sementara fungsi `aturPeluru` digunakan untuk mengatur gerakan masing-masing peluru yang terekam di *database* peluru. Kedua fungsi tersebut perlu ditulis, dan bisa diletakkan di bawah fungsi `gameLoop`.

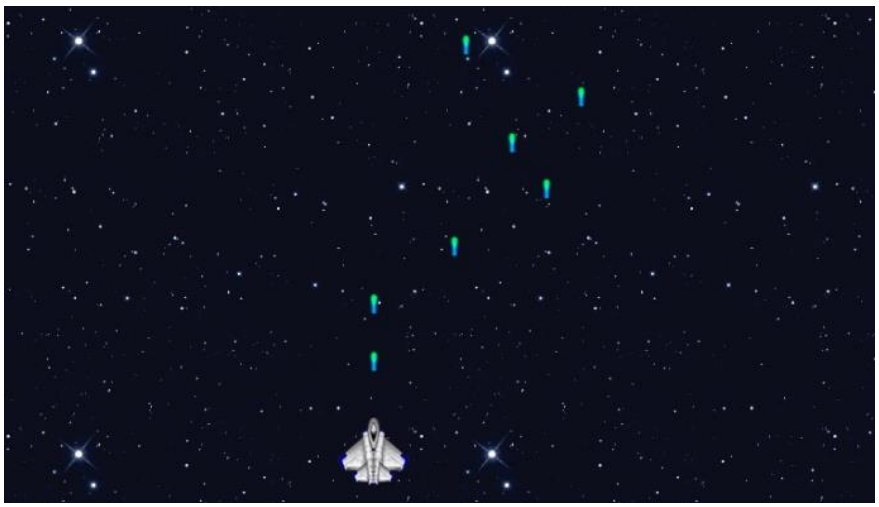
```
48. function gameLoop(){
49.         . . .

65. }
66.
67. function tembak(ob, kec){
68.         if (ob.waktuTembak == undefined || ob.waktuTembak == 0){
69.                 ob.waktuTembak = 10;
70.                 //tentukan jenis peluru
71.                 var peluru;
72.                 if (ob == game.jet){
73.                         peluru = setSprite(dataGambar.peluru1);
```

```
74.             }else{
75.                     peluru = setSprite(dataGambar.peluru2);
76.             }
77.             peluru.x = ob.x;
78.             peluru.y = ob.y;
79.             peluru.kec = kec;
80.             peluru.cek = 0;
81.             peluru.aktif = true;
82.             peluru.pemilik = ob;
83.             game.dataPeluru.push(peluru);
84.       }
85. }
86.
87. function aturPeluru(){
88.       if (game.jet.waktuTembak > 0) game.jet.waktuTembak--;
89.       for (var i = 0; i < game.dataPeluru.length;i++){
90.             var peluru = game.dataPeluru[i];
91.             peluru.y += peluru.kec;
92.             sprite(peluru);
93.             if (peluru.x < 0 || peluru.x > game.lebar || peluru.y < 0
                    || peluru.y > game.tinggi || !peluru.aktif){
94.                   game.dataPeluru.splice(i, 1);
95.             }
96.       }
97. }
```

Fungsi `tembak` memiliki 2 parameter yaitu `ob` atau objek yang akan dijadikan acuan peluru, dan parameter `kec` yaitu kecepatan peluru. Selanjutnya pada fungsi tersebut ditentukan sebuah *variabel* `game.waktuTembak = 10;` (baris 69) yang berfungsi untuk membatasi waktu keluarnya peluru. Tanpa pembatasan waktu, pemain dapat menahan tombol spasi dan peluru akan keluar secara terus menerus. Dengan menambah `waktuTembak` sebanyak 10 poin, maka peluang pemain menembakkan peluru adalah sebesar 30 (*fps*) dibagi 10 poin, yaitu 3 peluru per detik. Pada baris 71 dibuat *variabel* peluru, yang kemudian diatur segala propertinya, dan pada akhirnya dimasukkan (di*push*) ke dalam *Array* `game.dataPeluru` (baris 83).

Pada fungsi `aturPeluru`, digunakan operasi berulang `for` untuk mengeksekusi kode secara stimultan sejumlah data yang terekam di dalam variabel `game.dataPeluru`. Masing-masing peluru digerakkan pada sumbu Y sebesar variabel `kec` (kecepatan) dan kemudian digambarkan di *canvas* (baris 92). Pada baris 93 diberikan kondisi apabila peluru keluar dari area permainan atau peluru tidak aktif lagi `( !peluru.aktif` → tanda ! merupakan negasi

dari nilai *variabel*. Apabila *variabel* bernilai *true* (baris 81) , maka dengan adanya negasi kondisi tersebut dijalankan hanya ketika *variabel* `peluru.aktif` bernilai *false*). Untuk menghapus peluru dari *database*, data peluru dihapus dari *Array* `game.dataPeluru` dengan menggunakan kode `splice` (baris 94).



Gambar 36. Hasil penambahan fitur menembak

5.2 Menambahkan Musuh

Setelah fitur menembak ditambahkan, tahapan selanjutnya adalah menambahkan musuh. Berbeda dengan aset grafis pesawat atau aset grafis peluru, aset grafis untuk musuh dibuat dalam bentuk *spritesheet* atau beberapa gambar *sprite* yang dijadikan satu. Keuntungan penggunaan *spritesheet* ini adalah memperkecil ukuran *file* sehingga akan mempercepat proses *loading*. Masing-masing gambar musuh berukuran 64x64 *pixel* seperti halnya gambar pesawat, berjumlah 8 jenis gambar dan disatukan dalam satu *file* gambar berukuran 256x128 *pixel*.

Gambar 37. Aset grafis musuh dalam format *spritesheet* 256x128 *pixel*.

Setelah *file* gambar diletakkan ke dalam *folder* **assets**, maka perlu dilakukan identifikasi aset agar dapat dikenali oleh kode.

```
4.  var gambar = {
5.          . . .
6.          musuh:"musuh.png"
7.  }
```

Seperti halnya pengaturan peluru yang membutuhkan *varibel* bertipe *Array*, untuk mengatur pergerakan musuh juga diperlukan *variabel* untuk menampung data musuh. Oleh karena itu pada fungsi `setAwal` ditambahkan pengaturan variabel `dataMusuh`, sekaligus pengaturan jumlah musuh maksimal (`game.musuhMaks = 10`) dan kondisi jet pemain yang diset aktif pada awal permainan (`game.jet.aktif = true`). *Variabel* `jet.aktif` digunakan untuk memverifikasi prasyarat untuk keluarnya musuh, artinya selama jet pemain aktif maka musuh akan keluar secara acak.

```
41. function setAwal(){
42.         . . .

47.         game.dataMusuh = [];
48.         game.musuhMaks = 10;
49.         game.jet.aktif = true;
50. }
```

Pada fungsi `gameLoop` diperlukan kode untuk mengatur kemunculan dan gerakan musuh, sehingga disisipkan kode `aturMusuh()`.

```
52. function gameLoop(){
53.         ...

69.         aturMusuh();
70. }
```

Untuk mengatur musuh perlu dibuat fungsi `aturMusuh`. Fungsi ini dapat dituliskan di bagian paling bawah. Pada fungsi ini terdapat 2 hal, yaitu mengatur keluarnya musuh dan mengatur gerakan musuh. Perhatikan kode berikut :

```
104.        function aturMusuh(){
105.                //mengeluarkan musuh dengan sistem acak
106.                var musuh;
107.                if(game.dataMusuh.length < game.musuhMaks
                       && acak(100)==34 && game.jet.aktif){
108.                    musuh = setSprite(dataGambar.musuh, 64, 64);
109.                    musuh.y = -100;
110.                    musuh.x = 200+acak(game.lebar-400);
```

```
111.                    musuh.frame = acak(musuh.maxFrame)+1;
112.                    musuh.arah = 0;
113.                    musuh.kec = 2+acak(4);
114.                    musuh.aktif = true;
115.                    game.dataMusuh.push(musuh);
116.            }
117.            //gerakkan semua musuh
118.            for (var i = 0; i0 && acak(50) == 34) musuh.arah = 0;
119.                    //tembak
120.                    if (acak(100)==56)tembak(musuh, 10);

121.                    //tampilkan di layar
122.                    sprite(musuh);
123.                    //keluar dari layar
124.                    if (musuh.y > game.tinggi+100 || !musuh.aktif){
125.                            game.dataMusuh.splice(i, 1);
126.                    }
127.            }
128.    }
```

Pada baris 107 diberikan prasyarat untuk munculnya musuh, yaitu apabila musuh di layar masih kurang dari variabel musuhMaks (baris 48), jet pemain masih dalam kondisi aktif dan pengacakan. Pengacakan angka yang dimaksud digunakan untuk memberikan waktu keluarnya musuh, sehingga musuh tidak keluar secara terus menerus dan bersamaan.

```
acak(100)==34
```

Kode `acak(100)`, akan menghasilkan angka acak dengan rentang 0-99. Angka 34 sebenarnya adalah angka yang ditulis secara asal, kita boleh menuliskan angka berapapun antara 0-99. Kondisi tersebut berarti setiap langkah kode akan dilakukan pengacakan angka antara 0-99, dan jika hasil pengacakan tersebut sama dengan 34 maka kondisi akan terpenuhi. Hal ini berarti peluang munculnya musuh adalah 1 dibanding 99, meskipun demikian karena cepatnya proses yang dilakukan peluang munculnya musuh masih relatif tinggi. Metode pengacakan seperti ini umum digunakan dalam membuat kecerdasan buatan sederhana untuk musuh dalam game.

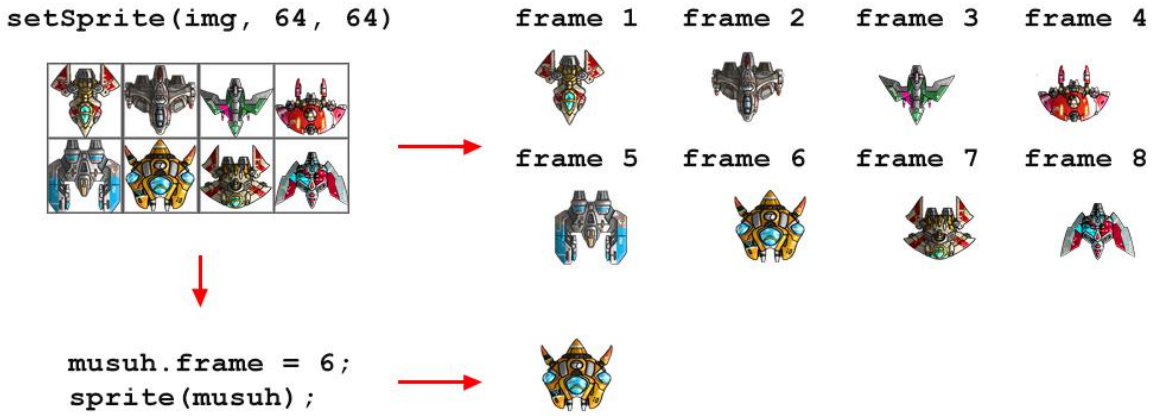
Metode lain yang dapat digunakan untuk mengatur keluarnya musuh adalah dengan menggunakan teknik rekaman data. Data waktu dan tempat keluarnya musuh direkam dalam *variabel* bertipe *Array*, dan ketika kondisi waktu terpenuhi, maka musuh akan keluar sesuai dengan data yang ada. Dengan metode ini musuh akan keluar dalam pola yang tetap, sehingga

game dapat diingat polanya oleh pemain. Metode ini akan dijelaskan pada bab berikutnya, yaitu pengaturan musuh pada game bertipe *platformer*.

Pada baris 108 dilakukan pengaturan *sprite* yang sedikit berbeda dengan *sprite* pesawat. Pada baris ini ditambahkan ukuran gambar yaitu 64 x 64 *pixel*.

```
musuh = setSprite(dataGambar.musuh, 64, 64);
```

fungsi `setSprite` memiliki parameter lebar dan tinggi gambar. Karena gambar musuh yang digunakan adalah *spritesheet* yang terdiri dari 8 gambar musuh, maka dengan mendefinisikan lebar dan tinggi gambar satuan (64 x 64 *pixel*), data yang terekam adalah data gambar satuan dalam format *frame*. Dalam kasus ini musuh memiliki 8 *frame* gambar, dan akan ditampilkan 1 *frame* saja sesuai dengan pengaturan kode. Perhatikan gambar berikut :



Gambar 38. Pengaturan *frame* pada *sprite*.

Pada baris 111 dilakukan pengacakan *frame* musuh dengan kode

```
musuh.frame = acak(musuh.maxFrame)+1;
```

sehingga variasi musuh yang muncul di layar dapat berbeda-beda sesuai hasil pengacakan angka. Selain itu kecepatan musuh juga diacak sehingga terdapat musuh yang bergerak cepat dan ada yang lebih lambat. Setelah *variabel* untuk pengaturan musuh telah diatur, maka musuh akan di*push* ke dalam *Array* `game.dataMusuh` untuk keperluan pengaturan selanjutnya.

Pada baris 118 dilakukan operasi berulang `for` sejumlah `game.dataMusuh.length`, yang di dalamnya terdapat pengaturan posisi, gerakan, serta penghapusan data dengan kode `splice` ketika musuh keluar dari layar atau musuh mati nantinya (`!musuh.aktif`).

Pada baris 134 ditambahkan kondisi pengacakan angka agar musuh menembakkan peluru ke arah bawah.

```
if (acak(100)==56)tembak(musuh, 10);
```

Pengacakan juga memiliki peluang 1 dibanding 99 untuk mengurangi jumlah peluru yang terlalu banyak di layar. Nilai kecepatan peluru adalah positif 10, sehingga peluru akan bergerak ke bawah, dan apabila mengacu pada kode baris 77 sampai 81 pada fungsi `tembak`, maka jenis peluru yang ditembakkan adalah peluru dengan *file* gambar **peluru2.png**.

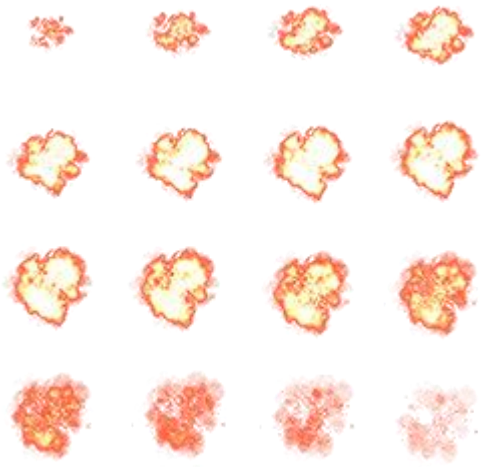
Gambar 39. Hasil penambahan musuh

5.3 Deteksi Tabrakan (*Collision detection*)

Setelah memahami proses menambahkan objek lain yang lebih kompleks ke dalam permainan, maka tahapan selanjutnya adalah melakukan deteksi tabrakan antara peluru dengan pesawat, atau tabrakan antara pesawat pemain dengan pesawat musuh. Di dalam *library* **gameLib.js** terdapat 2 metode deteksi tabrakan, yaitu dengan metode *bounding box*, dan metode *hitPoint*. Kedua metode tersebut dapat digunakan untuk mendeteksi tabrakan antara 2 objek atau antara objek dan kordinat tertentu (Ramtal dkk, 2014).

Sebagai tidak lanjut dari tabrakan akan ditampilkan animasi ledakan. Dalam artian ketika peluru pemain mengenai musuh, atau peluru musuh mengenai pemain, atau pemain tabrakan dengan musuh maka akan dimunculkan animasi ledakan. Untuk membuat animasi ledakan diperlukan sebuah *spritesheet* ledakan. Animasi ledakan dapat dibuat secara manual melalui aplikasi grafis atau melalui aplikasi *particle generator* yang banyak tersedia. Sebagai contoh dalam tutorial-1 ini digunakan *spritesheed* **ledakan.png** berukuran 256x256 *pixel* yang di

dalamnya terdapat 16 gambar animasi ledakan (masing-masing gambar memiliki ukuran 64 x 64 *pixel*).



Gambar 40. *Spritesheet* ledakan

Setelah *file* gambar diletakkan ke dalam *folder* **assets**, maka perlu dilakukan identifikasi aset agar dapat dikenali oleh kode.

```
4.  var gambar = {
5.          . . .
```

```
14.         ledakan:"ledakan.png"
15. }
```

Mengendalikan objek yang cukup banyak dan kompleks seperti gerakan musuh dan peluru lebih mudah apabila diatur dalam *variabel* bertipe *Array*, demikian pula dengan ledakan. Untuk mengatur posisi dan animasi ledakan dibutuhkan sebuah *variabel* yang diletakkan di fungsi `setAwal`.

```
42. function setAwal(){
43.         . . .
```

```
51.         game.dataLedakan = [];
52. }
```

Pengaturan tabrakan yang pertama adalah tabrakan antara peluru dan pesawat, baik itu pesawat musuh maupun pesawat pemain. Oleh karena itu perlu ditambahkan kondisi tabrakan dengan kode `hitPoint` sebagai berikut :

```
94. function aturPeluru(){
95.         if (game.jet.waktuTembak > 0) game.jet.waktuTembak--;
```

```
96.       for (var i = 0; i < game.dataPeluru.length;i++){
97.             . . .
```

```
103.            if (peluru.pemilik == game.jet){
104.                  //peluru pemain
105.                  peluru.cek++;
106.                  if (peluru.cek>game.dataMusuh.length-1)
                            peluru.cek = 0;
107.                  if (hitPoint(peluru.x, peluru.y,
                            game.dataMusuh[peluru.cek])){
108.                        game.dataMusuh[peluru.cek].aktif = false;
109.                        peluru.aktif = false;
110.                        ledakan(game.dataMusuh[peluru.cek].x,
                            game.dataMusuh[peluru.cek].y);
111.                  }
112.            }else{
113.                  //peluru musuh
114.                  if (hitPoint(peluru.x, peluru.y, game.jet)){
115.                        game.jet.aktif = false;
116.                        game.jet.mati = true;
117.                        peluru.aktif = false;
118.                        ledakan(game.jet.x, game.jet.y);
119.                  }
120.            }
121.      }
122.      aturLedakan();
123.   }
```

Penambahan pengecekan tabrakan peluru dibedakan berdasarkan opsi pemilik peluru (objek yang menembakkan peluru). Apabila peluru ditembakkan oleh pemain (baris 103), maka pengecekan terhadap musuh dilakukan secara bertahap. Pada baris 105 dilakukan penambahan secara bertahap pada *variabel* `peluru.cek`. Dalam tutorial lain, dapat ditemui perhitungan antara peluru dan musuh dengan operasi berulang `for`. Metode tersebut juga efektif namun membutuhkan memori yang besar karena terjadi operasi `for` bertingkat. Apabila komputer memiliki kecepatan yang kurang baik, proses menghitung tabrakan antar objek yang banyak akan memperlambat performa, sehingga pada tutorial ini digunakan metode perhitungan bertahap antara peluru dengan musuh satu persatu.

Untuk mendeteksi tabrakan antara peluru dan pesawat digunakan kode `hitPoint` sebagai berikut :

```
hitPoint(posisiX, posisiY, Objek)
```

ketika kondisi tabrakan terpenuhi, maka ditambahkan `ledakan` pada posisi objek (baris 110 dan 118). Setelah `ledakan` ditambahkan, objek perlu dinon-aktifkan menggunakan kode `objek.aktif = false;` atau `objek.mati = true;` Pada baris 122 ditambahkan kode `aturLedakan` untuk mengatur animasi ledakan. Oleh karena itu, fungsi `aturLedakan` perlu ditambahkan pada bagian akhir kode.

Selain deteksi tabrakan antara peluru dan pesawat, juga diperlukan deteksi tabrakan antara pesawat musuh dengan pesawat pemain. Untuk itu pada fungsi `aturMusuh` dilakukan penambahan kode deteksi tabrakan menggunakan kode `tabrakan`. Berbeda dengan kode `hitPoint` yang berdasarkan kordinat x dan y terhadap objek, kode `tabrakan` mendeteksi antara objek dengan objek lain dengan metode *bounding box* (berdasarkan tabrakan antar kotak).

```
125.      function aturMusuh(){
126.            for (var i = 0; i < game.dataMusuh.length; i++){
127.                  . . .

162.                  //tabrakan dengan pemain
163.                  if (tabrakan(musuh, game.jet)){
164.                        musuh.aktif = false;
165.                        ledakan(musuh.x, musuh.y);
166.                        game.jet.aktif = false;
167.                        game.jet.mati = true;
168.                        ledakan(game.jet.x, game.jet.y);
169.                  }
170.            }
171.      }
```

Pada kode deteksi `tabrakan` di atas, ketika kondisi `tabrakan` antara `musuh` dan `game.jet` terjadi maka kedua objek dinon-aktifkan dan diletakkan `ledakan` pada masing-masing koordinat objek.

Setelah memahami sistem tabrakan antar objek, selanjutnya perlu ditambahkan fungsi untuk menambahkan ledakan ke *canvas* dan fungsi untuk mengatur animasi ledakan. Prinsip dasar

meletakkan ledakan cukup sederhana, yaitu dengan menambahkan *sprite* ledakan ke kordinat objek yang meledak. Kemudian untuk mengatur animasi data ledakan harus di*push* ke dalam *variabel* `game.dataLedakan`.

Untuk mengatur ledakan pada fungsi `aturLedakan` prinsip mendasar yang harus dipahami adalah `sprite` akan diletakkan pada kordinat tempat ledakan, kemudian seiring waktu `frame` dari `sprite` akan ditambah, sehingga akan terbentuk animasi. Ketika `frame sprite` ledakan sampai pada `frame` terakhir, maka data ledakan akan dihapus. Apabila yang meledak adalah musuh, maka *score* akan ditambah, dan jika yang meledak adalah pemain, maka permainan akan diarahkan kembali ke `halamanCover`.

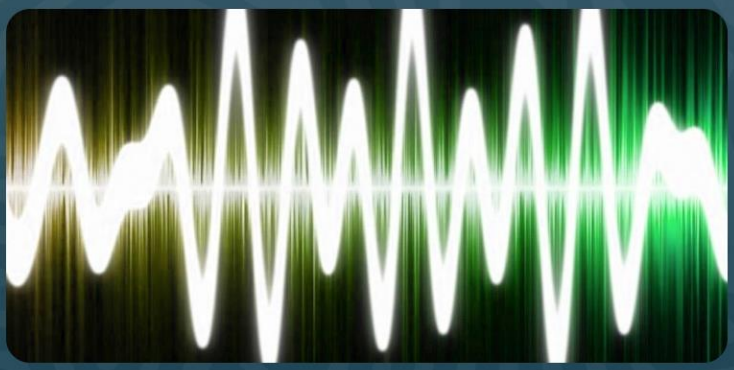
```
173.  function ledakan(px, py){
174.      var efekLedakan = setSprite(dataGambar.ledakan, 64, 64);
175.      efekLedakan.x = px;
176.      efekLedakan.y = py;
177.      efekLedakan.skalaX = (15+acak(5))/10;
178.      efekLedakan.skalaY = efekLedakan.skalaX;
179.      efekLedakan.rotasi = acak(360);
180.      game.dataLedakan.push(efekLedakan);
181.  }
182.
183.  function aturLedakan(){
184.      for (var i=0;i < game.dataLedakan.length;i++){
185.            var efekLedakan = game.dataLedakan[i];
186.            efekLedakan.frame++;
187.            if (efekLedakan.frame > efekLedakan.maxFrame) {
188.                  game.dataLedakan.splice(i, 1);
189.                  //dapat nilai atau game over
190.                  if (!game.jet.aktif){
191.                        jalankan(halamanCover);
192.                  }else{
193.                        tambahScore(10);
194.                  }
195.            }else{
196.                  sprite(efekLedakan);
197.            }
198.      }
199.  }
```

Sampai pada tahapan ini simpan *file* **game.js** dan jalankan *file* **index.html**. Apabila tidak ada kesalahan dalam penulisan kode, maka akan didapatkan sebuah *game* yang hampir utuh. Pemain sudah dapat menembak musuh dan meledakkannya, atau terkena serangan dan

permainan berakhir. Dengan penambahan sistem *score* dan musik, maka *game* ini sudah siap untuk dipublikasikan.

Gambar 41. Hasil penambahan deteksi tabrakan dan efek ledakan

Pada titik ini, pemahaman tentang mengelola beberapa objek melalui *variabel* bertipe *Array* dan menggerakkannya dengan operasi berulang `for` telah disampaikan. Hal ini menjadi dasar berikutnya dalam pengembangan sebuah *game*/permainan. Hal yang harus ditekankan dalam pengembangan *game* adalah proses pengembangan akan lebih mudah jika dilakukan secara bertahap. Setiap penambahan suatu sistem baru, kode harus diuji coba, dipastikan kode berjalan tanpa ada kesalahan, baru melangkah ke sistem berikutnya.



Sound Effect

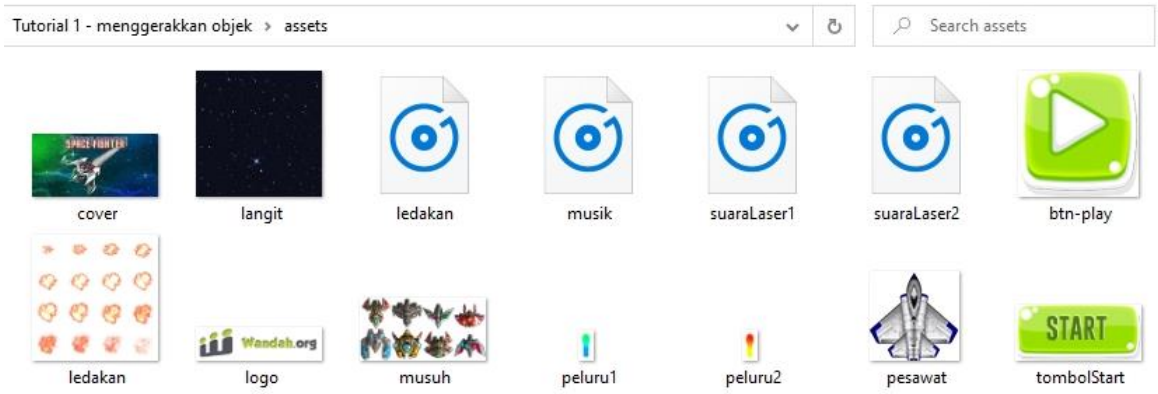
Penambahan efek suara (SFX) ke dalam game akan meningkatkan imersi. Permainan menjadi lebih realistik lebih menantang, dan kesenangan dalam bermain akan meningkat.

BAB 6
SUARA

6.1 Menambahkan Suara

Suara berfungsi untuk meningkatkan suasana permainan (*mood*) dan menghasilkan imersi dari kejadian yang dimunculkan di layar. Dengan keberadaan suara, *game* akan menjadi lebih menarik untuk dimainkan karena menjadi semakin realis. Bahkan beberapa latar musik dari suatu *game* bisa menjadi ikonik dan menjadi aspek yang dikenal oleh pemain game.

Dalam pengembangan *game*/aplikasi HTML5 mememilki prosedur yang hampir sama dengan penambahan gambar. *File* suara yang telah disiapkan diletakkan ke dalam *folder* **assets**, kemudian dilakukan identifikasi *file* dalam *variabel* `dataSuara`. Sebagai contoh pada tutorial-1 akan ditambahkan suara latar (*BGM*) dan suara efek menembakkan peluru serta ledakan (*SFX*). Adapun ekstensi *file* (tipe *file*) yang dapat digunakan sebagai *file* suara adalah *file* bertipe WAV dan MP3.



Gambar 42. Penambahan *file* suara pada *folder* assets.

Sedangkan untuk memainkan suara sebanyak satu kali digunakan kode :

```
mainkanSuara(dataSuara.nama);
```

sedangkan untuk memainkan suara sebagai suara latar (*BGM* / musik) digunakan kode :

```
musik(dataSuara.nama);
```

untuk mengatur volume suara dapat ditambahkan parameter volume, sebagai contoh apabila ingin suara bervolume 50%, maka kode yang digunakan adalah :

```
mainkanSuara(dataSuara.nama, 50);
```

Untuk menambahkan suara pada *game* pesawat pada **tutorial-1**, buka kembali *file* **game.js**, kemudian tambahkan kode untuk mengidentifikasi *file* suara yang akan digunakan sebagai berikut :

```
16. //file suara yang dipakai dalam game
17. var suara = {
18.         peluru1:"suaralaser1.mp3",
19.         peluru2:"suaralaser2.mp3",
20.         ledakan:"ledakan.mp3",
21.         bgm:"musik.mp3"
22. }
```

Setelah *file* suara diidentifikasi, maka *file* suara dapat mulai dimainkan. Seperti penjelasan pada bab sebelumnya terkait pengaturan keamanan yang melarang memainkan suara sebelum ada interaksi pemain dengan elemen HTML *canvas*, maka suara baru dapat ditambahkan pada `halamanCover`. `halamanCover` dijalankan setelah pemain melewati halaman `startScreen` dan telah berinteraksi dengan *canvas* melalui penekanan tombol `startBtn`, sehingga pada `halamanCover` dapat ditambahkan musik latar (*BGM*).

```
35. function halamanCover(){
36.         . . .

44.         musik(dataSuara.bgm);
45. }
```

Untuk memberikan suara efek (*SFX*) peluru ditambahkan 2 jenis suara, yaitu suara `peluru1` untuk peluru yang ditembakkan pemain dan suara `peluru2` yang ditembakkan musuh.

```
79. function tembak(ob, kec){
80.         if (ob.waktuTembak == undefined || ob.waktuTembak == 0){
81.                 ob.waktuTembak = 10;
82.                 //tentukan jenis peluru
83.                 var peluru;
84.                 if (ob == game.jet){
85.                         peluru = setSprite(dataGambar.peluru1);
```

```
86.                     mainkanSuara(dataSuara.peluru1);
87.             }else{
88.                     peluru = setSprite(dataGambar.peluru2);
89.                     mainkanSuara(dataSuara.peluru2);
90.             }

97.             . . .
98.     }
99. }
```

Sedangkan untuk memberikan suara efek (*SFX*) ledakan, ditambahkan suara `ledakan` pada fungsi `ledakan`.

```
180.     function ledakan(px, py){
181.             . . .

188.             mainkanSuara(dataSuara.ledakan);
189.     }
```

Simpan *file* **game.js** kemudian jalankan kembali *file* **index.html**. Pada tahapan ini permainan akan menjadi lebih menarik karena memiliki musik dan suara efek.

Antar Muka

Graphical User Interface (GUI/Antarmuka) memberikan informasi terkait game seperti score, giliran, energi dan sebagainya

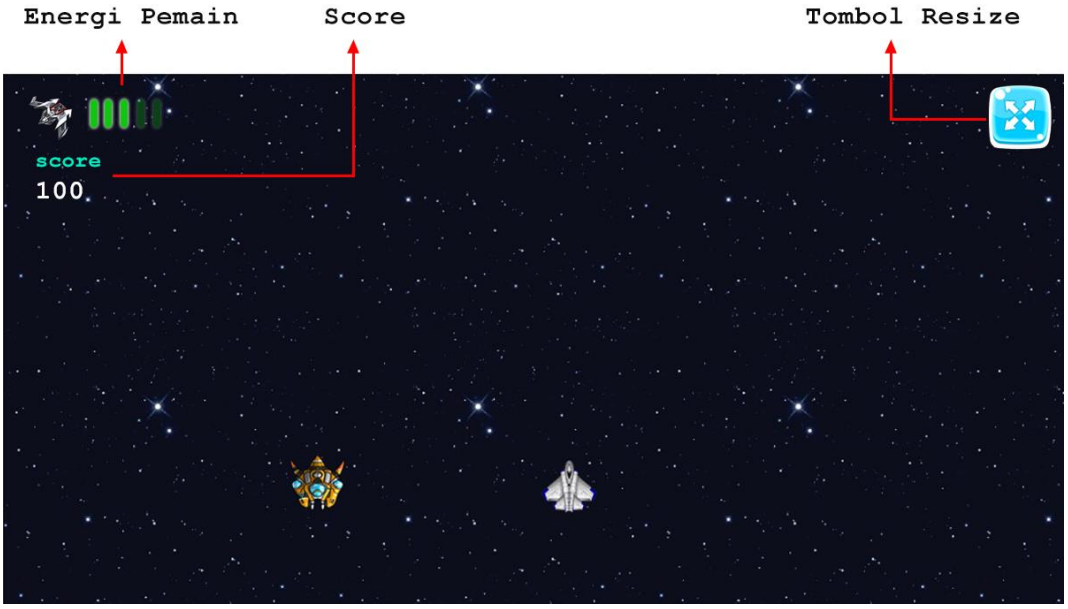
BAB 7

GRAPHICAL USER INTERFACE (*GUI* / ANTAR MUKA)

7.1 *Graphical User Interface* (*GUI*)

GUI merupakan antar muka pada sebuah aplikasi yang ditampilkan secara visual untuk mendukung interaksi antara pengguna dan aplikasi. Melalui *GUI* pemain dapat berinteraksi dengan aplikasi/*game* baik itu melalui informasi yang terbaca dari layar, menekan sebuah tombol, atau melakukan aktivitas lain yang akan mempengaruhi jalannya aplikasi/*game* (Fox, 2005).

Dalam *game* yang dibuat pada tutorial-1 misalnya, terdapat tombol untuk memulai permainan. Tombol tersebut merupakan salah satu implementasi dari penggunaan GUI dalam game. Namun demikian pada *game* pesawat pada **tutorial-1** pemain masih belum mendapatkan pengalaman yang baik terkait antar muka. Pemain belum dapat melihat score permainan, score tertinggi dan *game* yang tiba-tiba *game* over ketika pesawat meledak. Hal ini akan menyebabkan *game* dimainkan hanya beberapa kali saja, dan tidak lagi menarik untuk dimainkan. Oleh karena itu pada **tutorial-1** perlu ditambahkan beberapa aspek terkait dengan *GUI* agar menjadi lebih menarik.



Gambar 43. Rencana penambahan GUI

7.2 Tombol *resize*

Game HTML 5 pada umumnya ditujukan untuk *web browser* (web *game*), oleh karena itu untuk mendapatkan pengalaman bermain yang baik pada web *game* ditambahkan tombol *resize* untuk menjadikan mode layar penuh (*fullscreen*). *Library* **gameLib.js** memiliki fitur tombol *resize* namun harus ditambahkan 2 aset visual untuk tombol *resize*, yaitu tombol untuk *maximize* dan untuk *minimize*. Pada contoh ini digunakan 2 *file* dengan nama **maxBtn.png** dan **minBtn.png** yang selanjutnya diletakkan ke dalam *folder* **assets**.

Gambar 44. Tombol *resize*.

Agar kedua *file* tersebut dapat diidentifikasi oleh kode, maka diperlukan identifikasi pada *variabel* `gambar` sebagai berikut :

```
4.  var gambar = {
5.          . . .
```

```
16.         maxBtn:"maxBtn.png",
17.         minBtn:"minBtn.png"
18. }
```

Seperti halnya memainkan suara, untuk melakukan mode layar penuh (*fullscreen*) aplikasi HTML *canvas* harus berinteraksi terlebih dahulu dengan pemain (tidak dapat diaktifkan secara otomatis di saat aplikasi baru di mulai). Sehingga, peletakkan kode untuk *resize* baru dapat dilakukan pada fungsi `gameCover`. Selain itu tombol *resize* juga dapat diletakkan pada fungsi `gameLoop`.

```
37. function halamanCover(){
38.         . . .
```

```
47.         resizeBtn(1150,50);
48. }
49.

62. function gameLoop(){
63.         . . .
```

```
80.         resizeBtn(1150,50);
81. }
```

Dengan penambahan kode `resizeBtn(1150,50);` tombol *resize* akan diletakkan pada kordinat x = 1150 dan y=50 (di pojok kanan atas), dan apabila ditekan maka akan masuk ke mode *fullsceen* atau sebaliknya.

Gambar 45. Posisi tombol *resize*.

7.3 *GUI* energi dan score

Untuk menambahkan *GUI* yang menampilkan energi diperlukan 2 *file* gambar, yaitu *file* **icon-pesawat.png** (50x50 *pixel*) dan *file* **energi-bar** (16x40 *pixel*) yang selanjutnya diletakkan di dalam *folder* **assets**.

Gambar 46. Aset untuk GUI energi

Agar kedua *file* tersebut dapat diidentifikasi oleh kode, maka diperlukan identifikasi pada *variabel* `gambar` sebagai berikut :

```
4. var gambar = {
5.         . . .
```

```
17.         ikonEnergi:"icon-pesawat.png",
18.         barEnergi:"energi-bar.png"
19. }
```

Untuk menampilkan `score`, pada awal permainan perlu dilakukan pengaturan `score`. Selain itu untuk menambahkan energi pada pesawat pemain agar tidak meledak dalam satu kali tembak, maka diperlukan pengaturan pada fungsi `setAwal` sebagai berikut :

```
52. function setAwal(){
53.         . . .

62.         game.energi = 10;
63.         game.fullEnergi = 10;
64.         game.score = 0;
65. }
```

Untuk menampilkan *GUI* pada halaman permainan, pada fungsi `gameLoop` perlu ditambahkan kode sebagai berikut :

```
67. function gameLoop(){
68.         . . .

86.         gambarGUI();
87. }
```

Karena pesawat jet diset memiliki `energi` dan tidak mati dalam satu kali tembak atau satu kali tabrakan, maka pada fungsi `aturPeluru` dan `aturMusuh` perlu ditambahkan kondisi untuk mengurangi `energi` dan baru meledak ketika `energi = 0`.

```
111.        function aturPeluru(){
112.                . . .

130.                            //peluru musuh
131.                            if (hitPoint(peluru.x, peluru.y, game.jet)){
132.                                    peluru.aktif = false;
133.                                    game.energi--;
134.                                    if (game.energi == 0){
135.                                            game.jet.aktif = false;
136.                                            game.jet.mati = true;
137.                                            ledakan(game.jet.x, game.jet.y);
138.                                    }
139.                            }
140.                    . . .
141.        }

145.        function aturMusuh(){
146.                . . .
```

```
182.                    //tabrakan dengan pemain
183.                    if (tabrakan(musuh, game.jet)){
184.                           musuh.aktif = false;
185.                           ledakan(musuh.x, musuh.y);
186.                           game.energi--;
187.                           if (game.energi == 0){
188.                                  game.jet.aktif = false;
189.                                  game.jet.mati = true;
190.                                  ledakan(game.jet.x, game.jet.y);
191.                           }
192.                    }
193.             }
194.      }
```

Untuk menampilkan *GUI* pada fungsi `gameLoop` baris 86 dijalankan kode `gambarGUI`. Oleh karena itu perlu dibuat sebuah fungsi `gambarGUI` yang di dalamnya menampilkan `energi` pemain serta `score`. Untuk menampilkan `energi` pemain digunakan operasi berulang `for` sejumlah variabel `fullEnergi`. Penggunaan efek *alpha* (“`alpha=30`”) digunakan untuk menampilkan gambar secara transparan 30%, sehingga sisa `energi` akan terlihat lebih solid di layar. Sementara untuk menampilkan teks `score` cukup dengan kode `teks("teks yang ingin ditampilkan", kordinat x, kordinat y, pengaturan teks);`.

```
225.      function gambarGUI(){
226.             //energi bar
227.             tampilkanGambar(dataGambar.ikonEnergi, 50, 50);
228.             for (var i=0;i < game.fullEnergi;i++){
229.                    var stat = "";
230.                    if (i > game.energi) stat = "alpha=30";
231.                    tampilkanGambar(dataGambar.barEnergi, 100+i*15, 50,
   stat);
232.             }
233.             //tampilkan score
234.             teks("Score", 900, 30, "Calibri-bold-14pt-left-biru");
235.             teks(game.score, 900, 60, "Calibri-bold-20pt-left-putih");
236.             teks("Hi-score", 1000, 30, "Calibri-bold-14pt-left-kuning");
237.             teks(game.hiScore, 1000, 60, "Calibri-bold-20pt-left-
   putih");
238.      }
```

Gambar 47. Hasil fungsi `gambarGUI`.

7.4 *Virtual Joystick* dan Tombol *Mobile*

Game HTML 5 memungkinkan untuk diakses melalui gawai Android/iOS, namun ketika sebuah halaman web yang berisi aplikasi/*game* diakses melalui gawai *mobile*, permasalahnya adalah input pemain. Untuk memberikan input pemain harus menyentuh tombol virtual yang tampak di layar (McAllister dan White, 2015). *Library* **gameLib.js** telah dilengkapi dengan fitur input layar (*touch screen*), dan mendukung *joystick* virtual. Pada **tutorial-1** misalnya untuk menggerakkan pesawat diperlukan *joystick* sebagai pengganti tombol panah, dan diperlukan sebuah tombol untuk menggantikan tombol spasi. Untuk tombol pengganti spasi dibuat sebuah grafis lingkaran sederhana dan disimpan dengan nama **tombolA.png** dan diletakkan ke dalam *folder* **assets**.

Gambar 48. Tombol pengganti spasi pada mode layar sentuh

Untuk menggunakan gambar tersebut sebagai tombol virtual, terlebih dahulu gambar diidentifikasi melalui kode pada variabel gambar :

```
4.  var gambar = {
5.          . . .

19.         tombolA:"tombolA.png"
20. }
```

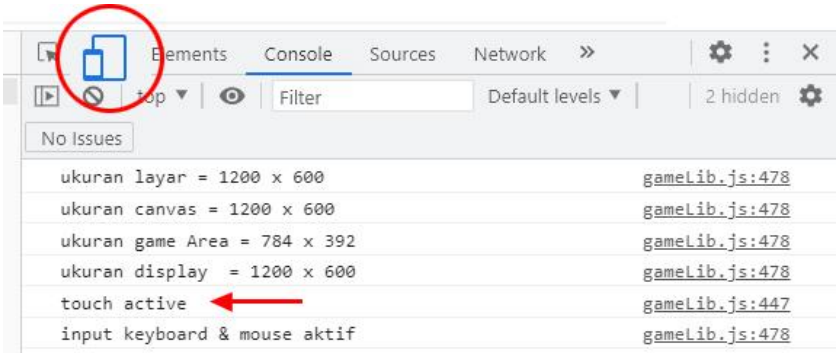
Untuk menampilkan tombol dan *joystick* virtual perlu diidentifikasi apakah *game* dimainkan pada perangkat *mobile*. Deteksi tersebut menggunakan kode `isMobile`, dan apabila bernilai `true`, maka tombol virtual digambarkan ke *canvas* serta ditambahkan *joystick* dengan kode `gambarJoyStick` pada fungsi `gameLoop`.

```
68. function gameLoop(){
69.         . . .
```

```
88.         //mobile
89.         if (isMobile){
90.                 var tombolA = tombol(dataGambar.tombolA, 1050, 450);
91.                 if (tekan(tombolA)) tembak(game.jet, -5);
92.                 gambarJoyStick();
93.         }
```

```
94. }
```

Untuk menguji coba *game* pada mode *mobile* tidak harus menggunakan perangkat/ gawai yang sesungguhnya namun cukum menjalankan *file* **index.html** pada *web browser*. Pada *web browser* Google Chrome misalnya, pada mode *inspect* dapat dipilih mode *device* (bisa dengan menekan ikon di toolbar atau menekan tombol *shortcut* **Ctrl + Shift + M**). Pastikan ikon *mobile* berwarna biru, kemudian tekan tombol *refresh* untuk membuka ulang *file* **index.html** pada mode *mobile*. Pada panel *console log* juga akan muncul teks `touch active` untuk menandakan bahwa *game* masuk ke mode *mobile* (*touch*).



Gambar 49. Mengatur mode *mobile* pada Google Chrome.

Pada mode *mobile*, apabila anda menyentuh layar sebelah kiri akan muncul *joystick* virtual untuk menggerakkan pesawat. Sedangkan ketika menekan tombol A pesawat akan menembak.

Gambar 50. Tampilan pada mode *mobile*.

Saat ini *game* pada **tutorial-1** sudah selesai dan dapat di*publish*. Namun anda dapat menambahkan fitur-fitur lain seperti bonus, bos musuh, atau level yang berbeda-beda.

[halaman ini sengaja dikosongkan]

Game Platform

Salah satu genre game yang menarik untuk dipelajari adalah game platform Kompleksitas desain level, pengaturan musuh dan bonus menjadi tantangan bagi pembelajar

BAB 8

GAME PLATFORMER

8.1 Konsep *Game Platformer*

Game platform (atau sering disederhanakan sebagai *game platformer* atau *game jump 'n' run*) adalah genre video *game* yang tujuan utamanya mengontrol karakter yang melompat atau memanjat di antara *platform* untuk mencapai tujuan tertentu. *Platformer* dicirikan oleh desain level yang menampilkan medan yang memiliki variasi tantangan dan menuntut kemampuan karakter pemain (seperti melompat dan memanjat) untuk menavigasi lingkungan pemain dan mencapai tujuan di setiap *level*. Sebagai contoh, banyak *game platformer* yang terkenal seperti Super Mario, Sonic, Metal Slug dan sebagainya.

Gambar 51. *Game platformer* Super Mario (*Copyright* Nintendo)

Prinsip dari sebuah *game platformer* adalah memiliki beberapa elemen dasar sebagai berikut:

1. Karakter pemain

   Karakter pemain dalam *game platformer* pada umumnya memiliki beberapa animasi gerakan seperti gerakan diam, berjalan atau berlari, melompat, terkena musuh, mati dan sebagainya. Karakter diatur memiliki kemampuan tertentu seperti dapat melompat tinggi, menginjak musuh, menembakkan peluru, dan beberapa kemampuan lainnya.

2. arena permainan (*Platform*).

   Arena permainan dibangun atas susunan gambar yang diatur sedemikian rupa, menyerupai *platform* yang tidak datar (bervariasi). Proses penyusunan gambar dalam *game platform* juga relatif bermacam-macam, dimana dapat digunakan teknik gambar utuh (*artbase*)maupun teknik pengubinan (*tiling*).

3. Tujuan (*objective*)

   Dalam *game platformer* tujuan permainan tidak hanya bersifat *survival* (bertahan selama mungkin), namun lebih ke satu titik lokasi tertentu pada *platform*. Tidak jarang karakter harus mengumpulkan bonus atau item tertentu sebelum mencapai tujuan tersebut.

4. Musuh

   Dalam beberapa *game platform*, musuh dapat dibuat dengan gerakan sederhana seperti gerakan satu arah sampai dengan gerakan kompleks seperti gerakan yang merespon posisi pemain.

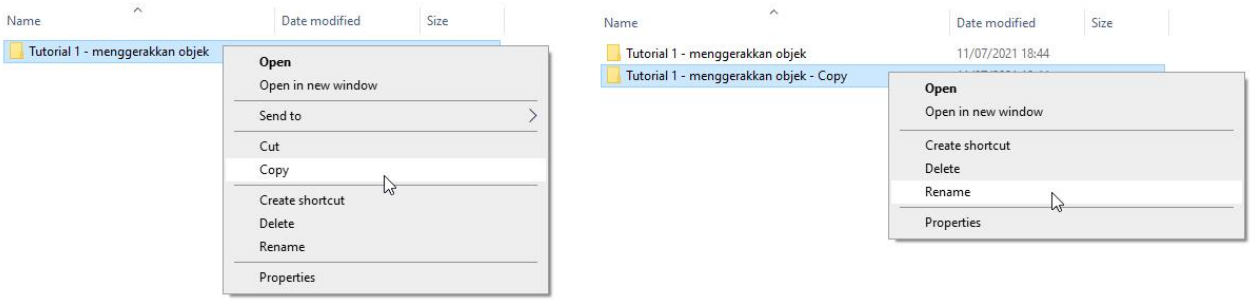
5. Item bonus

   Sering kali dalam *game platformer* pemain harus mengumpulkan koin atau bonus tertentu. Koin akan diakumulasikan dalam bentuk *score* untuk menentukan *score* tertinggi pemain. Selain itu pada *game-game* populer juga ditambahkan item bonus yang bersifat rahasia, untuk menambah daya tarik *game* dan membuat pemain mengulang-ulang permainan.

8.2 Membuat *project template Game* HTML 5

Sebelum melangkah ke tahapan membuat *game platformer*, terlebih dahulu akan dibuat sebuah *project template* untuk mempermudah proses pengembangan game. Dalam membuat *game* kita tidak harus memulai segalanya dari awal. Pada bab sebelumnya telah dibuat tutorial-1 yaitu *game* pesawat yang telah berjalan seutuhnya. *File* tersebut dapat dijadikan sebagai acuan dasar dalam pengembangan *game* HTML 5 atau bisa disebut sebagai *project template*. Perhatikan langkah berikut untuk membuat sebuah *project template*:

1. *Copy folder* tutorial-1 dengan cara klik kanan *folder* pada *windows explorer* kemudian pilih opsi *copy*. Kemudian klik kanan dan pilih opsi *paste*, sehingga kita memiliki 2 *folder* tutorial-1. Kemudian ubah nama *folder* hasil *copy* (*rename*) menjadi *folder* ***template***.



Gambar 52. Proses duplikasi *folder* proyek *game*

2. Buka pada *folder* **assets** dan hapus beberapa *file*. Sisakan *file* berikut :
   a. **btn-play.png** (gambar tombol untuk memulai permainan)
   b. **cover.jpg** (gambar pada halaman judul)
   c. **logo.png** (logo yang dimunculkan pada halaman *start*)
   d. **maxBtn.png** (gambar tombol untuk mode layar penuh)
   e. **minBtn.png** (gambar tombol untuk keluar dari mode layar penuh)
   f. **tombolStart.png** (gambar tombol pada halaman *start*)

Gambar 53. Aset gambar pada *folder* assets

3. Selanjutnya buka *folder* **js** dan buka *file* **game.js**. Pada tahapan ini akan dilakukan pengeditan kode agar *game* dapat dijalankan sampai dengan fungsi `gameLoop`, yang selanjutnya siap untuk digunakan sebagai *template* proyek *game* lainnya. Hapus beberapa baris kode, sehingga menjadi seperti pada kode berikut :

```
1. setGame("1200x600");
2. game.folder  = "assets";
3. //file gambar yang dipakai dalam game
4. var gambar = {
5.         logo:"logo.png",
6.         startBtn:"tombolStart.png",
7.         cover:"cover.jpg",
8.         playBtn:"btn-play.png",
9.         maxBtn:"maxBtn.png",
```

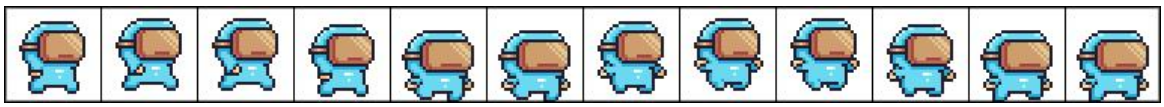
```
10.       minBtn:"minBtn.png"
11. }
12. //file suara yang dipakai dalam game
13. var suara = {
14. }
15.
16. //load gambar dan suara lalu jalankan startScreen
17. loading(gambar, suara, startScreen);
18.
19. function startScreen(){
20.       hapusLayar("#67d2d6");
21.       tampilkanGambar(dataGambar.logo, 600, 250);
22.       var startBtn = tombol(dataGambar.startBtn, 600, 350);
23.       if (tekan(startBtn)){
24.             jalankan(halamanCover);
25.       }
26. }
27. function halamanCover(){
28.       hapusLayar("#67d2d6");
29.       gambarFull(dataGambar.cover);
30.       var playBtn = tombol(dataGambar.playBtn, 1100, 500);
31.       if (tekan(playBtn)){
32.             setAwal();
33.             jalankan(gameLoop);
34.       }
35.       resizeBtn(1150,50);
36. }
37.
38. function setAwal(){
39. }
40.
41. function gameLoop(){
42.       hapusLayar();
43. }
```

5. Simpan *file* **game.js**, kemudian jalankan *file* **index.html** pada *web browser*. Pastikan halaman *start* dan halaman judul muncul, dan apabila tombol *play* ditekan maka akan dihasilkan layar kosong (akibat kode pada baris 42 fungsi `gameLoop`). Pada tahapan ini apabila tidak ada kesalahan, maka *folder* ***template*** akan siap digunakan untuk menjadi dasar dari *game-game* lainya.

8.3 Menggerakkan Karakter Pemain

Untuk membuat sebuah *game platformer*, tahapan pertama yang harus dipahami adalah menggerakkan pemain. Pada penjelasan di atas, sebuah karakter dalam *game platformer* harus dapat bergerak setidaknya dapat berjalan dan melompat. Gerakan tersebut membutuhkan gambar yang bergerak atau animasi. Dalam pengembangan *game* 2 dimensi khususnya *game* berbasis HTML5, animasi karakter dapat dibuat dalam format *spritesheet* .

Perhatikan gambar karakter berikut :

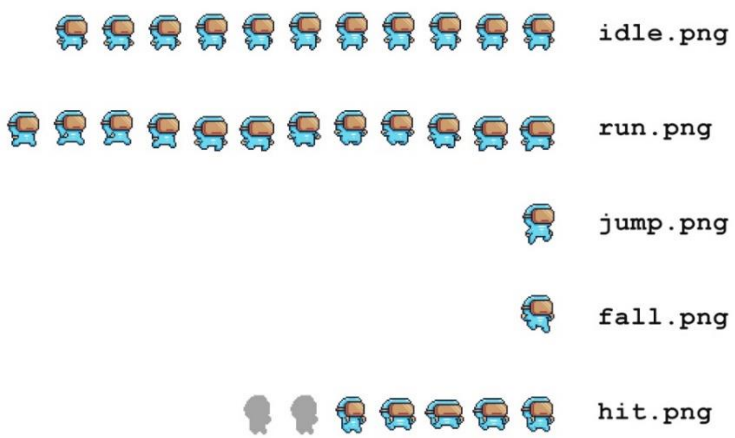


Gambar 54. *Spritesheet* karakter dengan animasi berlari

(Sumber : virtual guy, https://*pixel*frog-assets.itch.io/)

Pada gambar di atas, animasi karakter berlari dibuat secara *frame by frame* dengan ukuran 32x32 *pixel* sejumlah 12 gambar. Selanjutnya 12 gambar tersebut dijadikan 1 buah *file spritesheet* untuk mempermudah proses penganimasian, sekaligus untuk menghemat memori.

Dalam pengembangan *game* terkait dengan kebutuhan *spritesheet* diperlukan SDM yang secara khusus mampu mengembangkan grafis permainan. Alternatif lain yang dapat dilakukan adalah dengan memanfaatkan aset visual yang tersedia baik secara gratis maupun secara komersial pada beberapa situs penyedia aset game. Sebagai contoh dalam **tutorial-2** tentang pengembangan *game platformer* digunakan aset visual dari situs itch.io. Adapun gambar karakter yang digunakan untuk pembuatan *game platformer* ini adalah gambar karakter dengan animasi diam (*idle*), berlari, melompat, jatuh dan mati. *File* tersebut disusun dalam format *spritesheet* dan terpisah berdasarkan masing-masing animasi. Perhatikan gambar :



Gambar 55. Animasi karakter

Setelah *file* gambar karakter disiapkan, kita dapat memulai pembuatan *game platformer*. Perhatikan langkah berikut :

1. Copy *folder* ***template***, *paste* kemudian ubah nama *folder* hasil *copy-paste* menjadi *folder* **tutorial – 2**.

Gambar 56. Struktur *folder*

2. Buka *folder* **assets**. Khusus untuk *file* **cover.jpg** (yang digunakan pada halaman judul), perlu diganti dengan *file* baru. Sebagai contoh pada **tutorial-2** ini digunakan judul *game Momon Adventure* dengan gaya *pixel art*. *Copy file* cover baru, dan *paste*kan (*replace*) gambar cover lama.

Gambar 57. Cover baru untuk *game platformer Momon Adventure*

3. Selanjutnya letakkan *file* gambar animasi karakter (*spritesheet*) yang telah disiapkan sebelumnya (lihat gambar 48) ke dalam *folder* **assets**.

Gambar 58. *File* gambar pada *folder* **assets**

4. Setelah aset visual ditambahkan ke dalam *folder* **assets**, tahapan selanjutnya adalah mengedit kode **game.js**. Buka *file* **game.js** dan lakukan beberapa penambahan kode sebagai berikut :

Tahapan pertama adalah mengidentifikasi *file* gambar. Untuk itu edit variabel `gambar` dan tambahkan data gambar *spritesheet* karakter sebagai berikut :

```
4. var gambar = {
5.         logo:"logo.png",
6.         startBtn:"tombolStart.png",
7.         cover:"cover.jpg",
8.         playBtn:"btn-play.png",
9.         maxBtn:"maxBtn.png",
10.        minBtn:"minBtn.png",
11.        idle:"Idle.png",
12.        run:"Run.png",
13.        jump:"Jump.png",
14.        fall:"Fall.png"
15. }
```

Setelah data gambar teridentifikasi, tahapan selanjutnya adalah melakukan pengaturan awal pada fungsi `setAwal`. Pengaturan yang dimaksud adalah mengatur *sprite* karakter dan menentukan kordinat posisi karakter. Untuk itu tambahkan kode berikut pada fungsi `setAwal`.

```
43. function setAwal(){
44.        game.hero = setSprite(dataGambar.idle,32,32);
45.        game.hero.x = 800;
46.        game.hero.y = 268;
47.        game.skalaSprite = 2;
48.        game.lantai = 300;
49.        game.lompat = false;
50. }
```

Perhatikan kode di atas. Karakter dalam *game* yang akan dibuat disebut sebagai `game.hero` yang mana `game.hero` diset sebagai *sprite* dengan ukuran 32x32 *pixel* (baris 44). Pada baris 47 diatur *variabel* `game.skalaSprite = 2;` yang berfungsi untuk menampilkan gambar sebesar 200% atau 2 kali lipat. Sebagai catatan, *sprite* berukuran 32x32 *pixel* ketika ditampilkan pada *canvas* yang berdimensi 1200x600 *pixel* akan tampak terlalu kecil, sehingga perlu sedikit diperbesar. Pada baris 48 diatur lantai pada sumbu Y = 300, sehingga karakter nantinya akan berjalan pada garis lantai tersebut.

Pada baris 49 diatur variabel `game.lompat` yang digunakan untuk mengatur sistem gerak karakter nantinya. Setelah pengaturan awal selesai dilakukan, tahapan selanjutnya yang menjadi inti dari menggerakkan karakter permainan yaitu pada fungsi `gameLoop`. Tambahkan kode berikut pada fungsi `gameLoop`.

```
52. function gameLoop(){
53.         hapusLayar("#9c9695");
54.         garis(0, game.lantai, game.lebar, game.lantai);
55.         if (!game.lompat){
56.                 if (game.kanan){
57.                         game.hero.img = dataGambar.run;
58.                         game.hero.skalaX = 1;
59.                         game.hero.x+=3;
60.                 }else if (game.kiri){
61.                         game.hero.img = dataGambar.run;
62.                         game.hero.skalaX = -1;
63.                         game.hero.x-=3;
64.                 }else{
65.                         game.hero.img = dataGambar.idle;
66.                 }
67.                 if (game.atas){
68.                         game.lompat = true;
69.                         game.hero.img = dataGambar.jump;
70.                         game.lompatY = -10;
71.                 }
72.         }else{
73.                 game.lompatY+=0.5;
74.                 if (game.lompatY>0) game.hero.img = dataGambar.fall;
75.                 game.hero.y+=game.lompatY;
76.                 if (game.kanan){
77.                         game.hero.skalaX = 1;
78.                         game.hero.x+=3;
79.                 }else if (game.kiri){
80.                         game.hero.skalaX = -1;
81.                         game.hero.x-=3;
82.                 }
83.                 if (game.hero.y>=game.lantai-32){
84.                         game.hero.y = game.lantai-32;
85.                         game.lompat = false;
86.                 }
87.         }
88.         loopSprite(game.hero);
89. }
```

5. Simpan *file* **game.js** tersebut, kemudian jalankan *file* **index.html** pada *web browser*. Apabila tidak ada kesalahan, maka saat ini karakter dapat dikendalikan dengan tombol panah dan animasi karakter akan muncul sesuai dengan gerakan pemain.

Gambar 59. Hasil menggerakkan karakter

Penjelasan program

Pada baris 54 dibuat sebuah `garis` untuk mengidentifikasi lantai permainan. Pada tahapan selanjutnya garis ini akan diubah dengan gambar *platform* yang lebih kompleks. Selanjutnya dalam sebuah *game platform* terdapat 2 kondisi gerakan, yaitu gerakan saat karakter di udara (saat melompat) dan karakter saat di lantai. Kondisi ini perlu dipisahkan karena pada saat di lantai dan pemain tidak menekan tombol apapun, maka karakter akan berada pada posisi diam (*idle*), sementara pada saat lompat dan pemain tidak menekan tombol apapun, karakter tetap akan bergerak dan memiliki animasi yang berbeda saat karakter bergerak ke atas (melompat ke atas) dan saat karakter bergerak ke bawah (jatuh ke bawah).

Untuk mengatur gambar animasi yang dipakai, digunakan kode `game.hero.img`, sebagai contoh ketika tombol kanan ditekan (`game.kanan`) maka `game.hero.img` diset menjadi `dataGambar.run`, sehingga ketika kode `loopSprite(game.hero);` (baris 88) dijalankan, maka animasi yang muncul adalah animasi berlari.

Untuk gerakan berlari ke arah kiri, pada dasarnya digunakan gambar yang sama namun ditampilkan secara terbalik (refleksi pada sumbu x), dan untuk melakukannya digunakan kode `game.hero.skalaX = -1;` angka -1 tersebut merupakan pencerminan 100% terhadap sumbu x, sehingga gambar akan terlihat terbalik ke arah kiri.

Untuk melompat digunakan tombol panah atas, namun kondisi tersebut memiliki prasyarat yaitu karakter tidak dalam kondisi melompat (`!game.lompat`) jadi ketika karakter sedang berada di udara, karakter tidak dapat melompat lagi (panah atas tidak memberikan efek lagi). Ketika tombol panah atas ditekan (baris 67) dan karakter sedang berada di lantai, maka

variabel `game.lompat` akan bernilai *true* (baris 68) dan variabel `lompatY` akan bernilai negatif. Variabel `lompatY` merupakan kecepatan lompatan, yang akan terpengaruh sedikit demi sedikit oleh tarikan ke bawah (gravitasi) (lihat baris 73 `game.lompatY+=0.5;`).

Gambar 60. Proses karakter melompat

Seiring waktu, variabel `lompatY` dari yang bernilai negatif akan menjadi positif, sehingga ketika variabel `lompatY` ditambahkan pada kordinat Y karakter (baris 75), maka akan terjadi perubahan gerak ke atas kemudian turun ke bawah. Ketika kordinat Y karakter telah melewati garis `lantai` (baris 83) maka nilai `game.lompat` menjadi *false* kembali dan pemain bisa melompat kembali dengan menekan tombol panah atas. Konsep inilah yang menjadi dasar dari gerakan melompat sebuah karakter game.

8.4 Mendesain Arena Permainan (*Platform*)

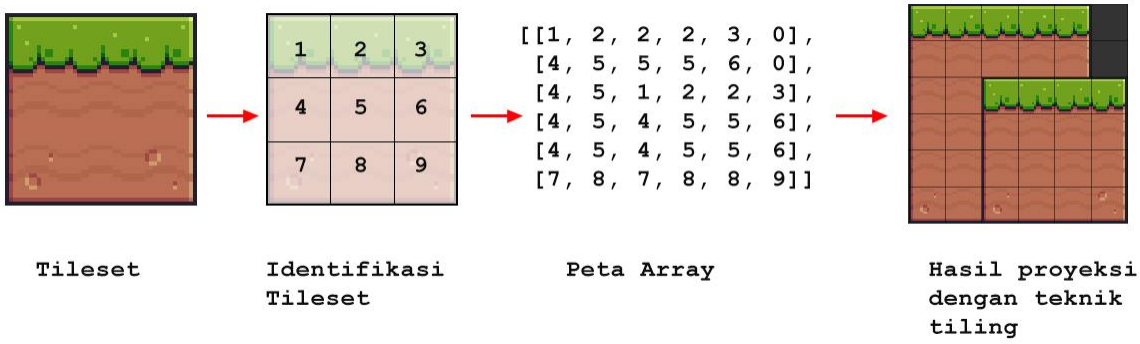
Sesuai namanya sebuah *game platformer* akan melibatkan *platform* atau arena permainan. Arena permainan akan memberikan rintangan tertentu kepada karakter pemain sampai karakter pemain tiba di tujuan. Dalam membuat arena permainan digunakan berbagai macam teknik seperti *art base* dan teknik *tiling* (Feil dan Scattergood, 2005). Teknik *art base* merupakan teknik mendesain arena permainan dengan gambar *platform* yang utuh, teknik ini menggunakan aset grafis yang tersusun atas gambar utuh. Sedangkan teknik *tiling* menggunakan gambar gambar kecil (*tile*) yang disusun secara berulang untuk membentuk satu kesatuan gambar.

Masing-masing teknik tersebut memiliki keunggulan. Teknik *art base* misalnya, desain arena bisa dibuat secara bebas menggunakan aplikasi grafis sehingga menghasilkan gambar yang sifatnya dinamis (organik). Namun teknik ini membutuhkan memori yang besar, serta sistem

perhitungan posisi objek yang lebih rumit. Sementara itu teknik *tiling* menggunakan perulangan-perulangan, sehingga akan membutuhkan memori yang lebih sedikit. Pada teknik *tiling* perhitungan posisi objek juga relatif lebih mudah, namun untuk mendesain level dengan teknik ini membutuhkan keterampilan dan kesabaran.

Gambar 61. Teknik *art base* pada *game* Rayman (kiri – *Copyright* Ubisoft) dan teknik *tiling* pada *game* Super Mario Bros (kanan – *Copyright* Nintendo)

Pada **tutorial-2** digunakan teknik tiling. Sistem *tiling* (pengulangan/pengubinan), merupakan sebuah metode untuk mengulang-ulang sebuah gambar berdasarkan data bertipe *array*, dimana dengan data *array* tersebut gambar sumber (disebut sebagai *tileset*) diolah dan diulang sedemikian rupa untuk digambar ulang menjadi satu gambar baru yang lebih besar. Perhatikan gambar berikut :

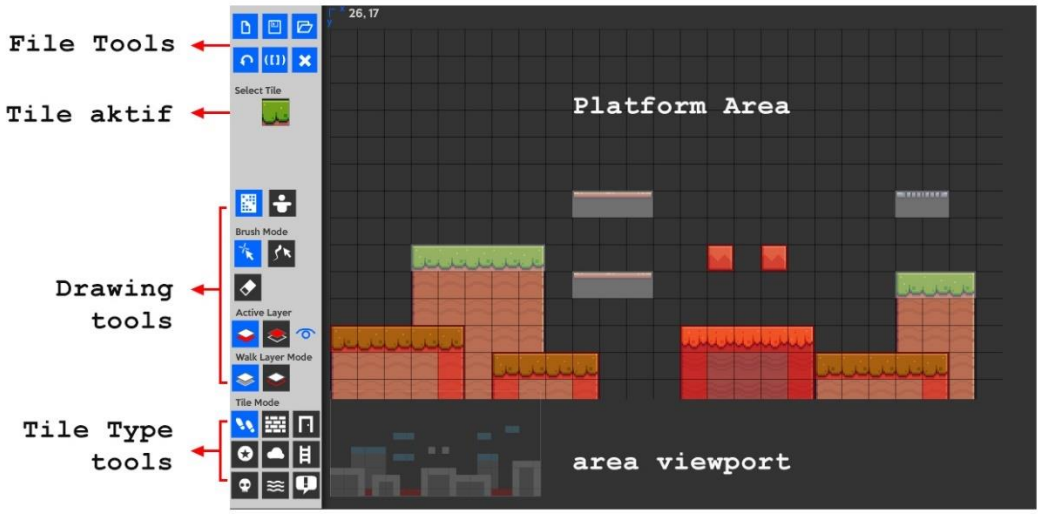


Gambar 62. Teknik *tiling*.

Pada metode *tiling* diperlukan sebuah *tileset* yang merupakan gambar sumber untuk menggambar keseluruhan arena permainan (Wibawanto, 2020). *Tileset* akan diidentifikasi berdasarkan lebar dan tinggi satu petak (satu *tile*). Selanjutnya dibutuhkan sebuah peta *Array* yang berisi kode angka yang digunakan untuk menyusun peta area permainan. Dengan operasi berulang `for`, kode angka dalam peta *array* tersebut diproyeksikan sesuai dengan identifikasi *tileset*, sehingga akan terbentuk sebuah gambar baru yang pada akhirnya digunakan sebagai arena permainan (*platform*) (Tyers, 2017).

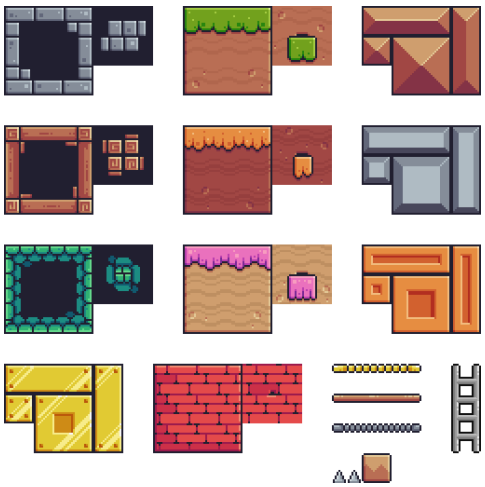
Mendesain area permainan dengan menyusun array akan menjadi rumit apabila dilakukan secara manual, oleh karena itu dibutuhkan sebuah editor khusus untuk menyusunnya. Pada **tutorial-2** digunakan aplikasi *tile editor* yang dapat diunduh pada tautan www.wandah.org/download/map-editor.exe

Setelah aplikasi ter*download*, aplikasi dapat secara langsung dijalankan dan akan didapatkan tampilan untuk mengedit arena permainan sebagai berikut :



Gambar 63. Tampilan aplikasi *map editor*.

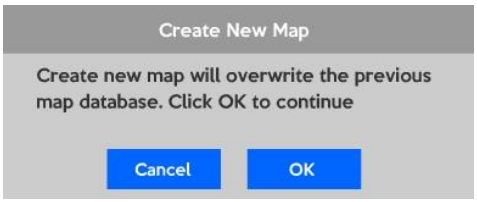
Untuk memulai membuat area permainan *game platformer* diperlukan sebuah *tileset* berukuran 512x512 *pixel* (pada saat buku ini ditulis, aplikasi *map editor* hanya dapat mengelola *tileset* berukuran 512x512 *pixel*). *Tileset* tersebut didesain sedemikian rupa agar memiliki *tile-tile* yang dibutuhkan dalam menyusun seluruh area permainan. Seperti halnya *spitesheet* karakter, *tileset* juga dapat dibuat sendiri atau memanfaatkan aset yang sudah tersedia baik secara gratis maupun secara komersial. Pada contoh *game platformer* tutorial-2 digunakan *tileset* gratis yang diunduh dari https://*pixel*frog-assets.itch.io/ sebagai berikut:



Gambar 64. *Tileset* **terain.png**

Untuk mulai membuat arena permainan, perhatikan langkah-langkah berikut :

1. klik tombol *New*. Langkah ini akan menghapus progress apapun yang sudah anda buat, sehingga pastikan anda sudah menyimpan *file* sebelum memulai membuat *file* baru.



Gambar 65. Membuat *file* baru

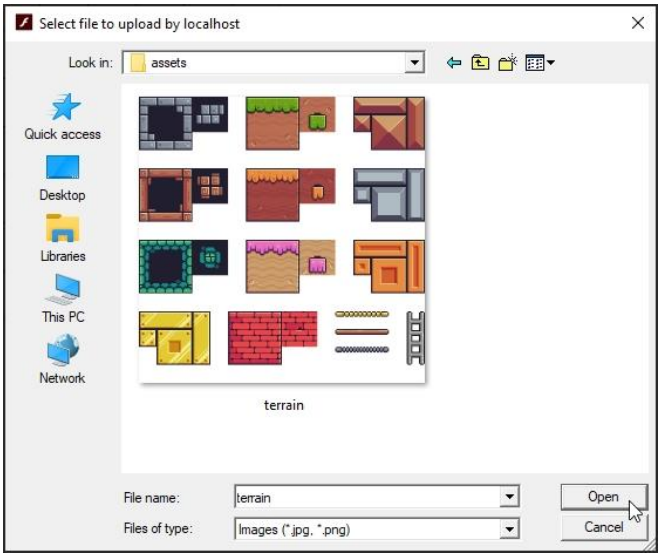
2. Kemudian atur ukuran area permainan, sebagai contoh untuk memahami konsep *tiling* cukup dibuat area yang tidak terlalu besar. Buatlah ukuran lebar 30 *tile* dan tinggi 15 *tile*. Salah satu tips dalam mengembangkan area *game platformer* adalah, diperlukan tinggi yang cukup (minimal 15 *tile*) untuk memberikan peluang pemain untuk melompat nantinya. Tinggi yang terlalu pendek akan menyebabkan gerakan pemain yang terbatasi.

   Pada bagian atas terdapat kolom *Map ID*. Variabel ini digunakan untuk membedakan masing-masing array. Sebagai contoh ketika kita memiliki 5 level, maka akan ada 5 peta *array* dan masing-masing harus memiliki nama variabel yang unik.

   Pada kolom *Map tileset* secara *default* terisi dengan angka 32. Angka tersebut merupakan ukuran dari masing-masing tile yaitu 32x32 *pixel*. Dalam standar pengembangan *game*, ukuran gambar pada umumnya menggunakan standar *binary* (*bits*), sehingga angka yang direkomendasikan adalah 16, 32, 64, 128 dan seterusnya.

Gambar 66. Mengatur *file* baru area permainan

3. Sebelum menekan tombol *OK*, perlu ditentukan terlebih dahulu *tileset* yang akan digunakan. Klik tombol *browse*, kemudian pilih *file* **terain.png** yang telah disiapkan sebelumnya.

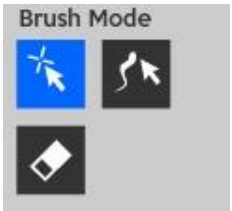


Gambar 67. Memilih *file tileset*.

4. Tekan *OK* dan area permainan siap untuk dibuat.
5. Tahapan awal dalam mendesain area permainan adalah meletakkan *tile* yang tepat pada arena. Untuk memilih *tile*, klik tombol *tile* aktif dan pilih *tile* pada *tileset*.

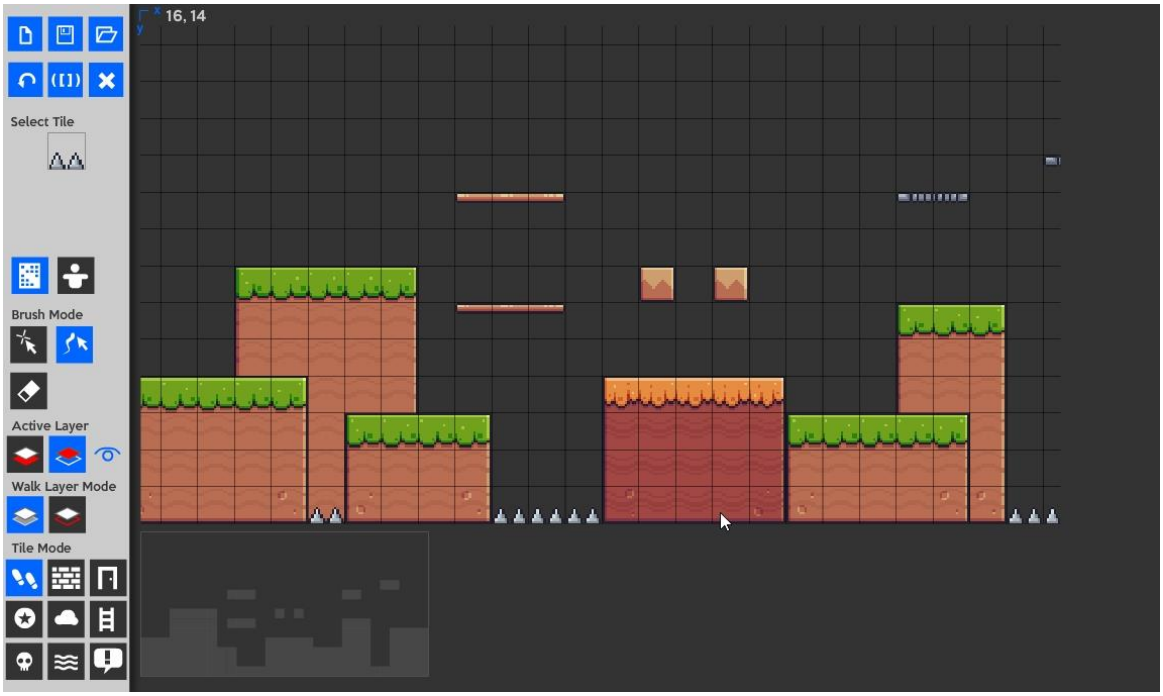
Gambar 68. Memilih *tile* aktif.

6. Terdapat 2 cara untuk meletakkan *tile* ke area permainan, yaitu dengan cara klik atau dengan cara *brush*. Cara *klik* dilakukan dengan menekan satu persatu, sementara *brush* mempercepat proses dengan menahan *mouse* (*drag*) dan menggerakkannya di sekitar area. Untuk menghapus *tile* dapat dilakukan dengan cara menekan tombol *undo* atau dengan memilih mode penghapus.



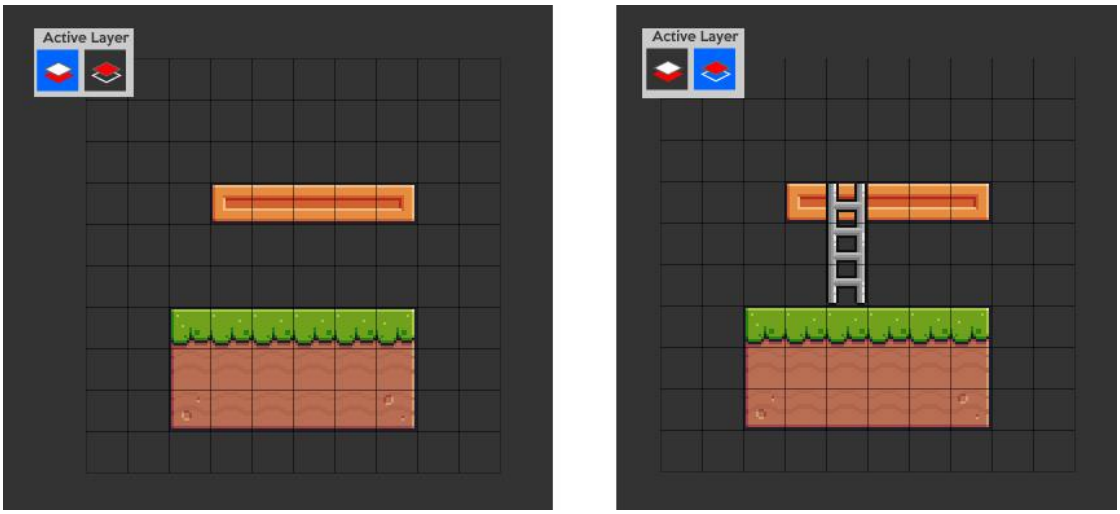
Gambar 69. Pilihan mode menggambar

7. Desain area sesuai dengan konsep yang diinginkan. Prinsip dasar dalam mengembangkan atau mendesain area permainan adalah area tersebut dapat dilalui oleh pemain. Dengan prinsip ini jarak lompatan akan dipertimbangkan. Kemampuan pemain dalam melompat (ke atas atau ke depan) harus disesuaikan. Menambahkan tantangan juga perlu disesuaikan dengan level permainan.



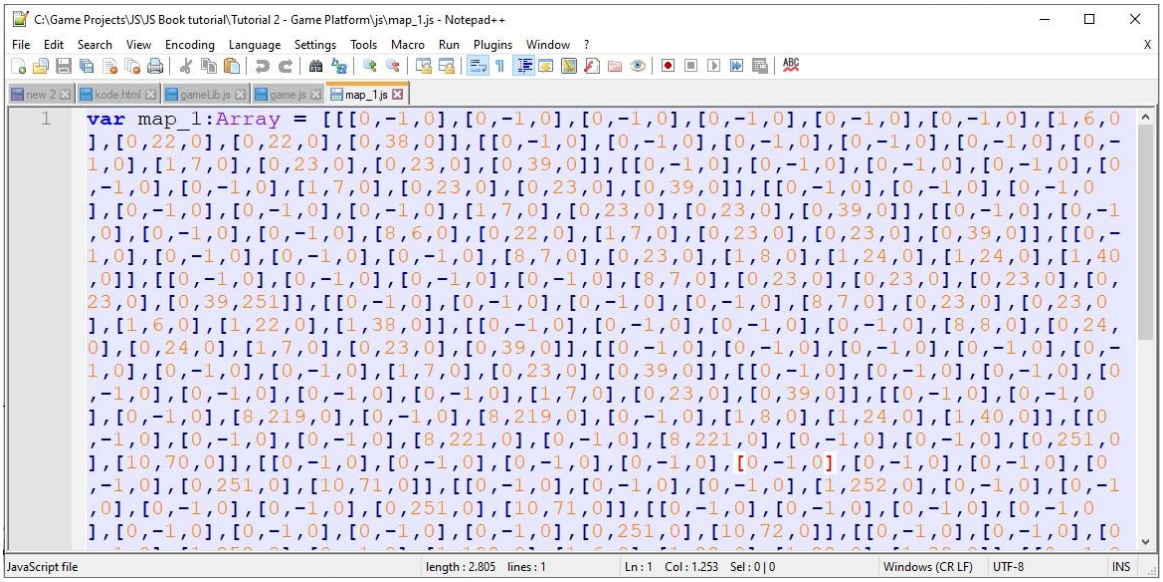
Gambar 70. Mendesain area permainan

8. Untuk membuat desain dengan *tile* yang saling menumpuk (*overlaping*), dapat memanfaatkan *layer* atas. Aplikasi *map editor* memiliki 2 lapisan (*layer*), secara *default layer* dasar (*layer* bawah) yang aktif. Untuk membuat gambar di atas gambar yang sudah ada dapat mengaktifkan mode *layer* aktif atas dan kemudian menekan kembali area yang dimaksud.



Gambar 71. Mode *layer* untuk menggambar secara menumpuk

9. Setiap kali selesai membuat satu bagian area permaian, anda dapat menyimpan *progres* dengan menekan tombol *save*. Pilih *folder* **js** kemudian simpan dengan nama **map_1.js**. Apabila anda membuka *file* **map_1.js** tersebut dengan menggunakan Notepad++, maka akan didapatkan kode *array* yang cukup rumit sebagai berikut:

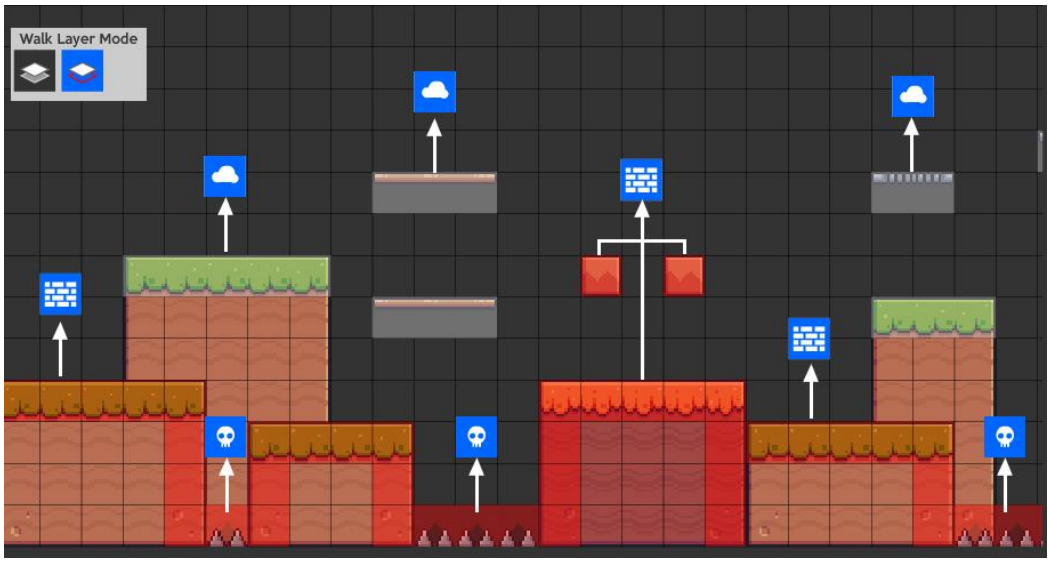


Gambar 72. Struktur *array* pada *file* **map_1.js**.

10. Tahap berikutnya adalah menambahkan tipe *tile*. Dalam *game* yang menggunakan teknik *tiling* masing-masing *tile* dapat diset memiliki tipe tertentu. Tipe *tile* secara mendasar dapat dibedakan menjadi beberapa, yaitu :
    a. *Walkable* (normal), yaitu *tile* yang dapat dilalui oleh pemain. Secara default seluruh *tile* yang dibuat di atas bersifat *walkable*.
    b. *Wall* (dinding), yaitu *tile* yang tidak dapat dilalui oleh pemain. Tile tipe ini digunakan sebagai lantai *platform* atau tembok penghalang.
    c. *Door* (pintu), yaitu *tile* yang menjadi pintu masuk untuk berpindah ke area lainnya. Sebagai contoh beberapa pipa dalam *game* Super Mario bisa dimasuki dan membawa ke area yang berbeda.
    d. *Item* (bonus item), yaitu *tile* yang memberikan bonus item seperti koin, bonus barang dan sebagainya.
    e. *Cloud / jump through* (awan), yaitu *tile* yang dapat ditembus dari bawah (saat pemain melompat), namun tidak tertembus dari atas atau dengan kata lain pemain bisa mendarat di atasnya.
    f. *Stair* (tangga), yaitu *tile* untuk menyusuri tangga.
    g. *Dead/trap* (jebakan), yaitu *tile* yang menjadikan karakter mati atau terluka saat melewatinya.

h. *Elemental* (air dan sejenisnya), yaitu *tile* yang memiliki sifat elemen yang diwakilinya. Sebagai contoh apabila tipe *tile* adalah air, maka pemain dapat mengapung atau berenang.
i. *Trigger* (sensor), yaitu tile yang akan memicu sesuatu hal, bisa jadi menginisiasi cerita dalam permainan, atau membuka syarat level tertentu.
j. Dan tipe-tipe lainnya.

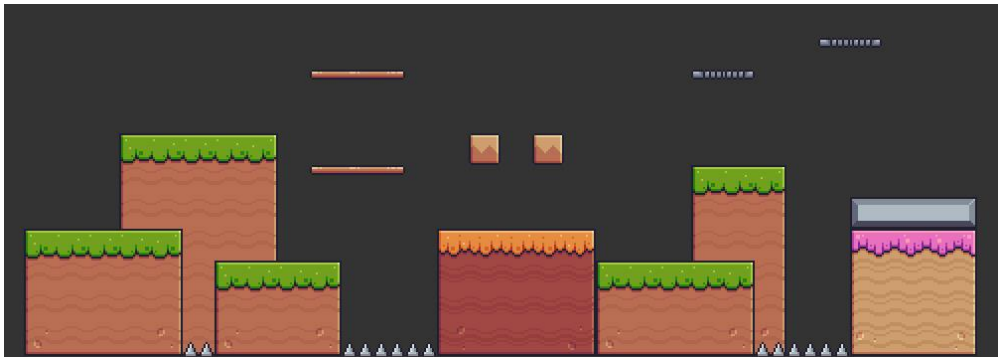
Untuk mengatur tipe *tile* pada *map editor* dapat digunakan mode *walk layer mode*, sehingga ketika mengedit tipe *tile*, gambar yang sudah dibuat sebelumnya tidak terganggu. Aktifkan *mode walk layer*, pilih jenis *tile* dan tambahkan jenis *tile* ke arena permainan.



Gambar 73. Menambahkan tipe *tile*.

11. Simpan kembali *file* **map_1.js**.

Pada tahapan ini desain *platform* yang dibuat masih relatif sederhana. Hal ini bertujuan untuk memahami konsep dasar gerakan karakter dalam area *platform*. Pada tahapan lanjutan, anda dapat mengembangkan desain arena permainan yang jauh lebih baik dan lebih kompleks.



Gambar 74. Desain arena permainan pada *file* **map_1.js**.

8.5 Menggerakkan Karakter di Arena Permainan (*Platform*)

Setelah arena permainan (*platform*) selesai dibuat dalam format peta *array* **map_1.js**, tahapan selanjutnya adalah menjadikan peta *array* tersebut sebagai arena permainan. Tahapan awal adalah menambahkan aset gambar *tile* set ke dalam *folder* **assets**, selain itu juga ditambahkan satu gambar sederhana yang digunakan sebagai latar (*background*).

Gambar 75. *File tileset* (**terrain.png**) dan *file* latar (**bg.jpg**)

Setelah gambar ditambahkan ke dalam *folder* **assets**, maka diperlukan identifikasi data gambar. Buka *file* **game.js**, kemudian lakukan penambahan kode sebagai berikut :

```
4.  var gambar = {
5.          . . .
```

```
16.         hit:"hit.png",
17.         tileset:"terrain.png",
18.         bg:"bg.png"
19. }
```

Berbeda dengan tutorial menggerakkan karakter pada sub bab sebelumnya (sub bab 8.3), untuk menggerakkan karakter pada arena permainan (*platform*) diperlukan pengaturan dasar *platform* serta animasi yang digunakan. Oleh karena itu fungsi `setAwal` perlu diubah menjadi sebagai berikut :

```
46. function setAwal(){
47.         game.hero = setSprite(dataGambar.idle,32,32);
48.         game.hero.animDiam = dataGambar.idle;
49.         game.hero.animJalan = dataGambar.run;
50.         game.hero.animLompat = dataGambar.jump;
51.         game.hero.animJatuh = dataGambar.fall;
52.         game.hero.animMati = dataGambar.hit;
53.         game.skalaSprite = 2;
54.         setPlatform(map_1, dataGambar.tileset, 32, game.hero);
55.         game.gameOver = ulangiPermainan;
56. }
```

Pada fungsi `setAwal` di atas, perlu dilakukan pengaturan karakter (disebut sebagai `game.hero`). Selanjutnya animasi masing-masing gerakan didefinisikan sesuai dengan *spritesheet* yang digunakan. Animasi yang digunakan adalah animasi diam, jalan, lompat, jatuh dan ditambahkan satu animasi hit (lihat baris 16 pada pengaturan variabel `gambar`) yang belum digunakan pada tutorial sebelumnya.

Pada baris 54 dilakukan pengaturan *platform* menggunakan kode:

```
setPlatform(map_1, dataGambar.tileset, 32, game.hero);
```

Parameter pertama adalah data *array* peta `map_1` yang telah dibuah dalam format *file* **map_1.js**. Parameter kedua adalah data gambar *tileset* yang akan digunakan untuk menyusun arena permainan. Parameter ketiga adalah ukuran *tile*, yang dalam hal ini adalah 32 *pixel*. Parameter terakhir adalah karakter yang akan digerakkan di arena permainan, yang dalam hal ini adalah `game.hero`.

Pada baris 55 ditambahkan kode `game.gameOver = ulangiPermainan;` yang berfungsi untuk mengatur permainan ketika karakter mati. Pada kode tersebut terdapat kode `ulangiPermainan` yang merupakan nama fungsi yang belum dibuat. Oleh karena itu perlu ditambahkan fungsi baru yaitu fungsi `ulangiPermainan` sebagai berikut :

```
58. function ulangiPermainan(){
59.         game.aktif = true;
60.         setAwal();
61.         jalankan(gameLoop);
62. }
```

Pada dasarnya fungsi di atas sangat sederhana, untuk mengulangi permainan cukup dilakukan pengaturan awal kembali dengan kode `setAwal` dan menjalankan fungsi `gameLoop`. Pada baris 59 variabel `game.aktif` diset bernilai *true*, hal ini dikarenakan ketika karakter pemain mati, variabel `game.aktif` akan bernilai *false* dan harus diset ulang ketika permainan dimulai lagi.

Untuk menggerakkan karakter pada area permainan dilakukan pada fungsi `gameLoop`. Berbeda dengan kode tutorial sebelumnya (sub bab 8.3) pada fungsi `gameLoop` pengaturan gerakan di arena *platform* menggunakan kode yang lebih sederhana, sebagai berikut:

```
26. function gameLoop(){
27.         hapusLayar("#9c9695");
28.         if (game.kanan){
29.               gerakLevel(game.hero, 3, 0);
30.         }else if (game.kiri){
31.               gerakLevel(game.hero, -3, 0);
32.         }
33.         if (game.atas){
34.               gerakLevel(game.hero, 0, -10);
35.         }
36.
37.         latar(dataGambar.bg, 0, 0.5);
38.         buatLevel();
39. }
```

Untuk menggerakkan karakter pada *platform* digunakan kode

```
gerakLevel(karakter, kecepatanX, kecepatanY);
```

Apabila anda membuka *file* **gameLib.js** dan melihat struktur fungsi `gerakLevel`, maka akan diketahui bahwa pergerakan objek pada *platform* dihitung berdasarkan peta *array* yang digunakan sebagai penyusun *platform*. Fungsi secara otomatis akan menggeser area permainan seiring gerakan karakter.

Pada baris 37 digunakan kode `latar(dataGambar.bg, 0, 0.5);` untuk menambahkan gambar latar dengan efek gerakan secara vertikal ke bawah. Pada baris 38 untuk menampilkan area permainan ke *canvas* digunakan kode `buatLevel();` maka area permainan akan digambar melalui operasi `for loop` berulang.

Simpan *file* **game.js**, namun sebelum menjalankan *file* **index.html** perlu dilakukan penambahan kode pada *file* **index.html** untuk membuka *file* **map_1.js**. Untuk itu buka *file* **index.html** dengan aplikasi Notepad ++ dan tambahkan satu baris kode untuk membuka *file* **map_1.js**.

```
20. <html>
21. <head>
22. . . .

12. </head>
13. <body>
```

```
14.      . . .

17. </body>
18. <script type="text/javascript" src="js/map_1.js"></script>
19. <script type="text/javascript" src="js/game.js"></script>
20. </html>
```

Simpan dan jalankan *file* **index.html** melalui *web browser*. Apabila tidak ada kesalahan, maka pada saat ini karakter *game* dapat bergerak pada area *platform*. Lakukan uji coba bergerak, melompat, dan menguji apakah karakter dapat mati dan *game* dapat diulang.

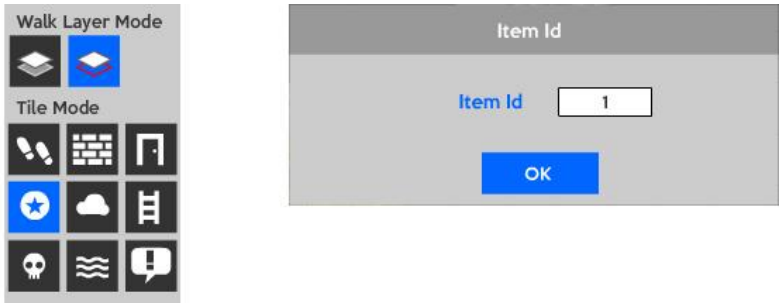
Gambar 76. Hasil kode menggerakkan karakter pada arena permainan

8.6 Menambahkan bonus item

Untuk menambahkan bonus, peta *array* pada *file* **map_1.js** harus di edit terlebih dahulu. Buka aplikasi *map editor* kemudian klik tombol *Open*. Pilih *file* **map_1.js**, kemudian pilih *open*. Sebelum menekan tombol OK, *tileset* perlu diatur. Klik tombol *browse* dan buka *file* **terrain.png** pada *folder* **assets**.

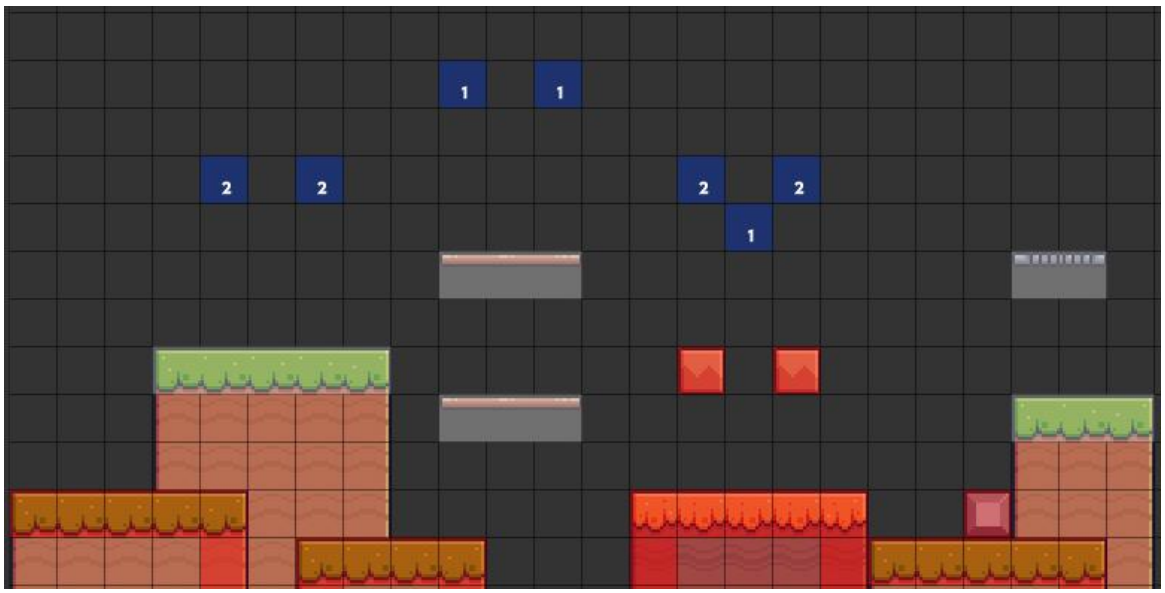
Gambar 77. Membuka kembali *file* **map_1.js**

Untuk menambahkan bonus, pilih mode *walk layer mode* dan tekan tombol *item*. Pada menu *popup*, ketikan *item ID* 1, kemudian tekan OK. *Item ID* merupakan identitas dari item yang akan ditambahkan, sebagai contoh akan ada 2 jenis bonus, maka *item ID* nantinya akan membedakan kedua jenis bonus tersebut.



Gambar 78. Mengatur *editor* untuk menambahkan bonus item

Tambahkan *tile* bonus sesuai dengan konsep yang diinginkan. Dalam mengedit selalu pertimbangkan jangkauan karakter pemain. Dalam hal ini kita dapat menyimpan *file*, menjalankan kembali *file* **index.html** dan mencoba apakah area yang dibuat memungkinkan untuk dimainkan atau tidak.



Gambar 79. Menambahkan *tile* item

Untuk menampilkan gambar bonus, terlebih dahulu perlu disiapkan gambar/*spritesheet* . Sebagai contoh pada **tutorial-2** ini akan digunakan 2 gambar bonus, yaitu gambar strawberi dan gambar buah kiwi. *File* yang digunakan sebagai item dapat berupa satu buah gambar (*sprite*) maupun gambar beranimasi (*spritesheet*). Kedua *file* yang akan digunakan di dalam contoh merupakan *spritesheet* dengan ukuran masing-masing *sprite* 32x32 *pixel*.

Gambar 80. *File* **strawberry.png** dan *file* **kiwi.png** sebagai *spritesheet* bonus

Letakkan kedua *file* tersebut ke dalam *folder* **assets**, kemudian buka *file* **game.js** dan lakukan identifikasi gambar sebagai berikut :

```
4.  var gambar = {
5.          . . .

18.         item1:"Strawberry.png",
19.         item2:"Kiwi.png"
20. }
```

Pengaturan item dilakukan pada fungsi `setAwal` menggunakan kode `setPlatformItem` sebagai berikut :

```
48. function setAwal(){
49.         . . .

58.         //set item
59.         setPlatformItem(1, dataGambar.item1);
60.         setPlatformItem(2, dataGambar.item2);
61. }
```

Dengan pengaturan tersebut, maka setiap *tile* item akan berubah gambarnya sesuai dengan pengaturan. Sebagai contoh seluruh *tile* item dengan item ID = 1 akan berubah menjadi gambar strawberi (`dataGambar.item1`).

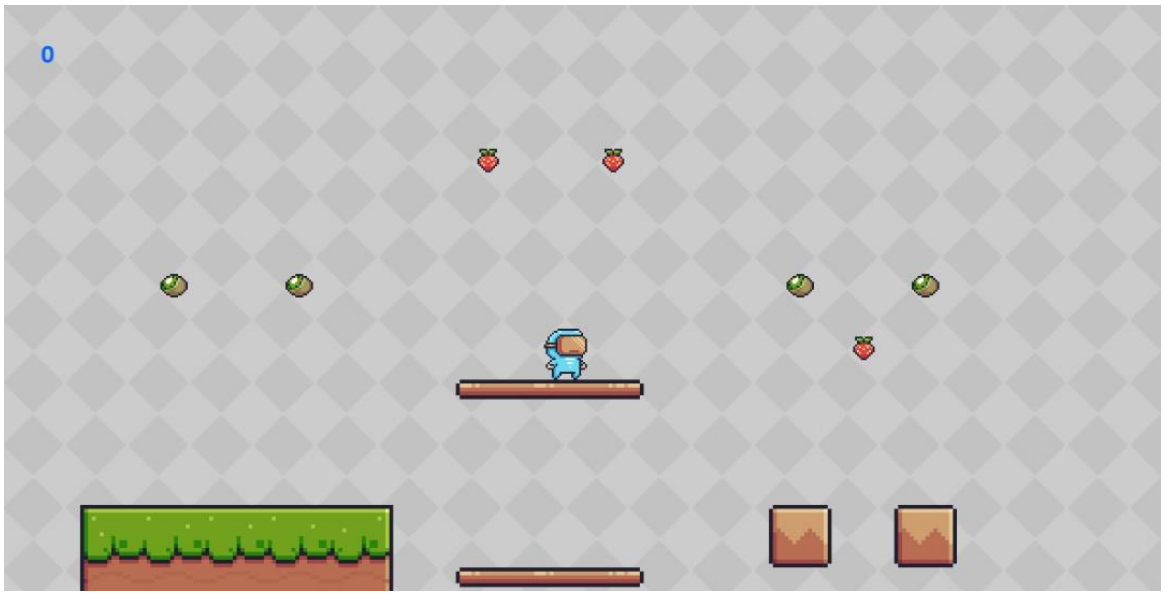
Pada saat ini apabila *file* **game.js** anda simpan, dan *file* **index.html** dijalankan maka bonus item akan muncul. Namun ketika diambil belum menghasilkan apapun, hanya menghilang dari layar. Untuk mendeteksi didapatkannya item dapat digunakan kode `game.itemID`, sebagai contoh ketika pemain mendapatkan item strawberi maka variabel `game.itemID` akan bernilai 1. Hal ini dapat digunakan untuk mengidentifikasi item yang didapatkan pemain dan dapat digunakan untuk mengatur score. Sebagai contoh implementasi, pada fungsi `gameLoop` tambahkan kode berikut :

```
69. function gameLoop(){
70.         . . .
71.         cekItem();
72.         teks(game.score, 40, 60, "Calibri-bold-20pt-left-biru");
73. }
```

Kode `cekItem` pada baris 71 merupakan fungsi yang belum dibuat, dan akan digunakan untuk mengidentifikasi item apa yang didapatkan oleh pemain. Sementara pada baris 72 kode `teks` digunakan untuk menampilkan score di pojok kiri atas. Selanjutnya pada bagian akhir baris perlu ditambahkan fungsi `cekItem` sebagai berikut :

```
86. function cekItem(){
87.         if (game.itemID > 0){
88.                 tambahScore(10*game.itemID);
89.                 game.itemID = 0;
90.         }
91. }
```

Simpan *file* **game.js** dan jalankan *file* **index.html** maka permainan saat ini menjadi lebih menarik dengan adanya bonus item.



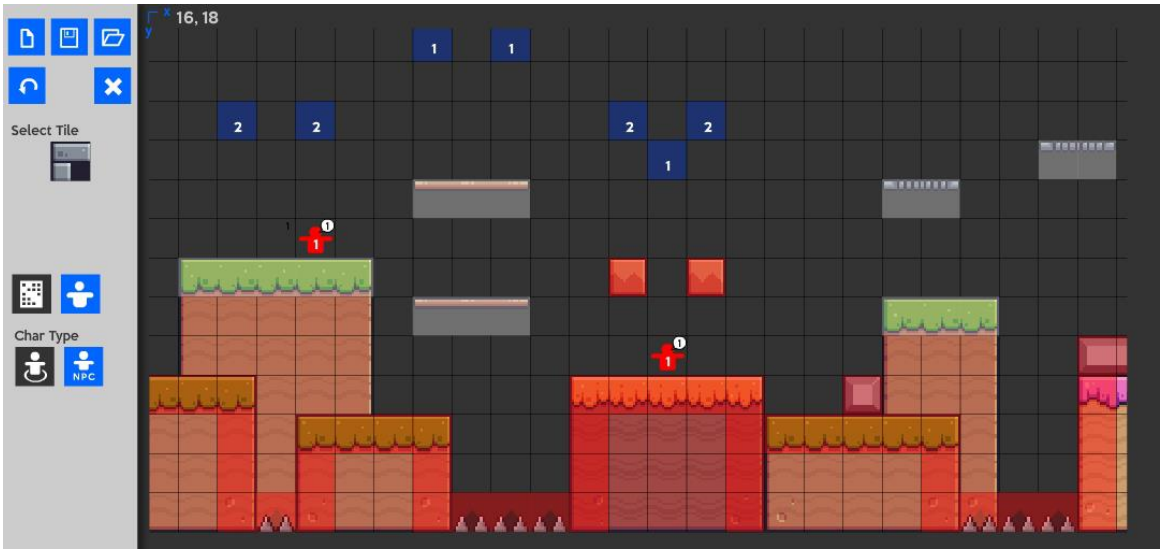
Gambar 81. Hasil penambahan bonus item

8.7 Menambahkan Musuh

Musuh pada *game platform* merupakan salah satu elemen penting yang membuat *game* menjadi lebih menantang. Seperti halnya menambahkan bonus item, untuk menambahkan musuh, peta *array* pada *file* **map_1.js** harus di edit terlebih dahulu melalui aplikasi *map editor*. Buka kembali *file* **map_1.js** kemudian pilih menu karakter dan tekan tombol *NPC* (*Non Playable Character*).

Gambar 82. Mengatur musuh

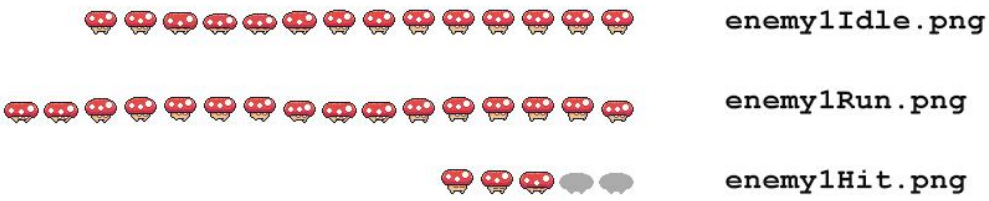
Seperti halnya pengaturan *item id*, pada panel pengaturan *NPC* terdapat *NPC id*. Id ini akan digunakan sebagai identifikasi musuh, sementara untuk kolom *Talk id* biarkan bernilai 1, karena kolom ini secara khusus ditujukan untuk fitur dialog pada *game* bertipe *Role Playing Game* (RPG). Setelah pengaturan selesai, klik OK dan tambahkan musuh ke arena permainan. Perhatikan posisi musuh dan potensi area gerak musuh. Sebagai contoh pada `map_1` ditambahkan 2 buah musuh dengan *NPC id* 1.



Gambar 83. Menambahkan musuh pada *map editor*

Untuk keperluan aset gambar musuh pada **tutorial-2** ini digunakan 3 buah *spritesheet* karakter jamur. *Spritesheet* tersebut adalah animasi karakter musuh saat diam (*idle*), animasi berjalan dan animasi saat diinjak oleh pemain. Pada pengembangan selanjutnya anda dapat

menambahkan gerakan lainnya pada musuh seperti gerakan melompat, menyerang, menembakkan peluru dan sebagainya.



Gambar 84. *Spritesheet* musuh

Setelah *spritesheet* siap dan diletakkan ke dalam *folder* **assets** langkah selanjutnya adalah mengidentifikasi gambar. Buka *file* **game.js** dan tambahkan kode berikut :

```
4.  var gambar = {
5.          . . .
```

```
20.         musuh1Idle:"enemy1Idle.png",
21.         musuh1Run:"enemy1Run.png",
22.         musuh1Hit:"enemy1Hit.png"
23. }
```

Untuk mengaktivasi musuh diperlukan pengaturan grafis menggunakan kode:

```
setPlatformEnemy(id musuh, objek);
```

Kode tersebut diletakkan pada saat pengaturan di awal permainan, dimana `id musuh` adalah *NPC id* yang ditambahkan pada area permainan, sedangkan `objek` merupakan variabel yang berisi tentang *spritesheet* animasi musuh. Tambahkan kode pada fungsi `setAwal` sebagai berikut :

```
50. function setAwal(){
51.         . . .
```

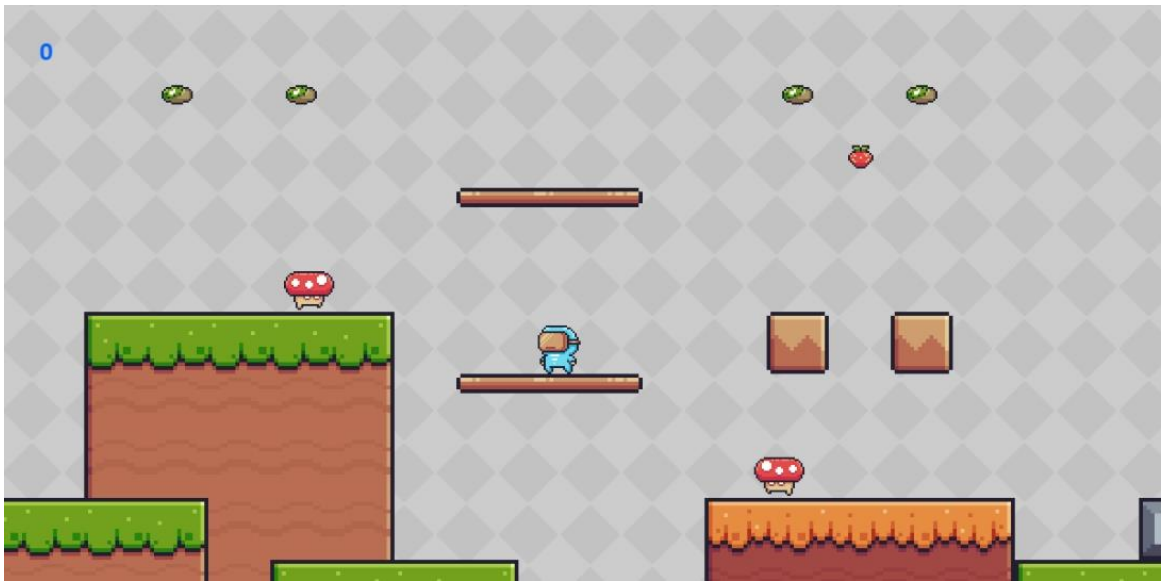
```
63.         //set musuh
64.         var musuh1 = {};
65.         musuh1.animDiam = dataGambar.musuh1Idle;
66.         musuh1.animJalan = dataGambar.musuh1Run;
67.         musuh1.animMati = dataGambar.musuh1Hit;
68.         setPlatformEnemy(1, musuh1);
69. }
```

Kode `setPlatformEnemy` secara otomatis akan mengatur musuh dan menampilkannya di arena permainan. Pada tutorial ini tipe musuh yang ditampilkan masih sangat sederhana, yaitu musuh yang bergerak pada *platform* ke kanan dan ke kiri. Karakter pemain akan mati apabila menyentuh musuh dan musuh akan mati apabila terinjak oleh pemain. Seluruh pengaturan gerakan musuh terdapat pada fungsi `gerakMusuh` yang terdapat di dalam *file* **gameLib.js**. Anda dapat mengembangkan jenis musuh yang lainnya dengan mengadaptasi dan menambah fitur pada fungsi tersebut. Sementara untuk mendeteksi musuh yang mati, dapat digunakan variabel `game.musuhID`. Sebagai contoh untuk menambah score saat musuh mati, dapat ditambahkan kode pada fungsi `cekItem` sebagai berikut :

```
94. function cekItem(){
95.         . . .
```

```
99.             if (game.musuhID != 0){
100.                    tambahScore(25);
101.                    game.musuhID = 0;
102.            }
103.    }
```

Simpan *file* **game.js**, kemudian jalankan kembali *file* **index.html** pada *web browser*. Apabila tidak ada kesalahan pengaturan, maka saat ini permainan akan menampilkan musuh pada area dan musuh dapat bergerak.



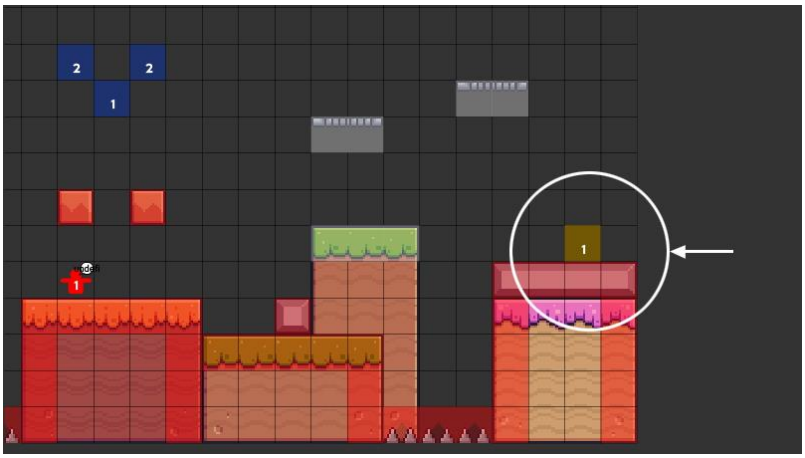
Gambar 85. Hasil penambahan musuh

8.8 Menambahkan Tujuan

Sebuah level pada *game platform* akan berakhir di satu titik tertentu sebelum berpindah ke level selanjutnya. Untuk menambahkan fitur tersebut pada dasarnya memiliki proses yang sama dengan menambahkan item atau menambahkan musuh. Langkah pertama adalah menambahkan *tile* dengan tipe *trigger* pada peta *array* sebagai penanda bahwa pemain telah sampai pada tujuan. Untuk melakukannya edit kembali **map_1.js** dan pilih mode tile *trigger*, lalu masukkan angka 1 pada *trigger id* dan klik OK.

Gambar 86. Mengatur mode *tile trigger*.

Trigger pada umumnya diletakkan di bagian terakhir dari arena permainan, namun anda dapat meletakkannya pada lokasi manapun yang dapat diakses oleh karakter pemain.



Gambar 87. Peletakan *tile trigger* pada akhir arena permainan.

Sebagai indikator *trigger* akhir permainan digunakan gambar *spritesheet* bendera (**Flag.png**) yang selanjutnya perlu diletakkan ke dalam *folder* **assets** dan dilakukan identikasi pada kode **game.js**.

Gambar 88. *Spitesheet* untuk indikator tujuan di akhir level.

```
4.  var gambar = {
5.          . . .
```

```
23.         bendera:"Flag.png"
24. }
```

Untuk menambahkan gambar bendera tersebut ke dalam arena permainan perlu dilakukan penambahan kode pada fungsi `setAwal` sebagai berikut :

```
51. function setAwal(){
52.         . . .
```

```
70.         //set trigger
71.         setPlatformTrigger(1, dataGambar.bendera);
72. }
```

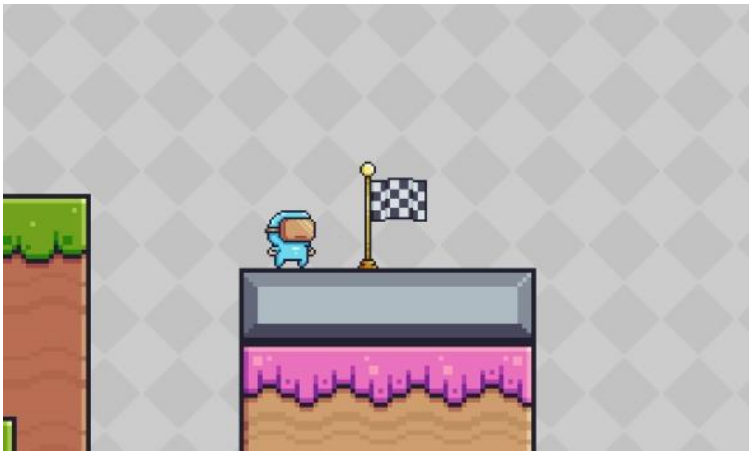
Untuk mengidentifikasi apakah pemain telah sampai pada *trigger* digunakan variabel `game.triggerID`. Identifikasi dapat diletakkan pada fungsi `cekItem` sebagai berikut:

```
97. function cekItem(){
98.         . . .
```

```
106.            if (game.triggerID == 1){
107.                    game.triggerID = 0;
108.                    game.aktif = false;
109.                    setTimeout(ulangiPermainan, 2000);
110.            }
111.        }
```

Pada kode di atas, ketika pemain menyentuh *trigger* dan variabel `game.triggerID` bernilai 1, maka *game* di non aktifkan sehingga pemain tidak bisa bergerak. Selanjutnya fungsi `ulangiPermainan` akan dijalankan setelah 2000 ms dengan kode `setTimeOut`. Jeda waktu tersebut dimaksudkan agar pemain mengetahui “proses kemenangan”. Apabila kode `ulangiPermainan` langsung dieksekusi saat pemain menyentuh *trigger* maka proses akan berlangsung dengan sangat cepat dan dalam dunia pengembangan *game* hal tersebut menghasilkan pengalaman bermain yang kurang baik.

Simpan *file* **game.js** dan jalankan kembali *file* **index.html** maka permainan akan berulang ketika pemain berhasil menyentuh bendera.



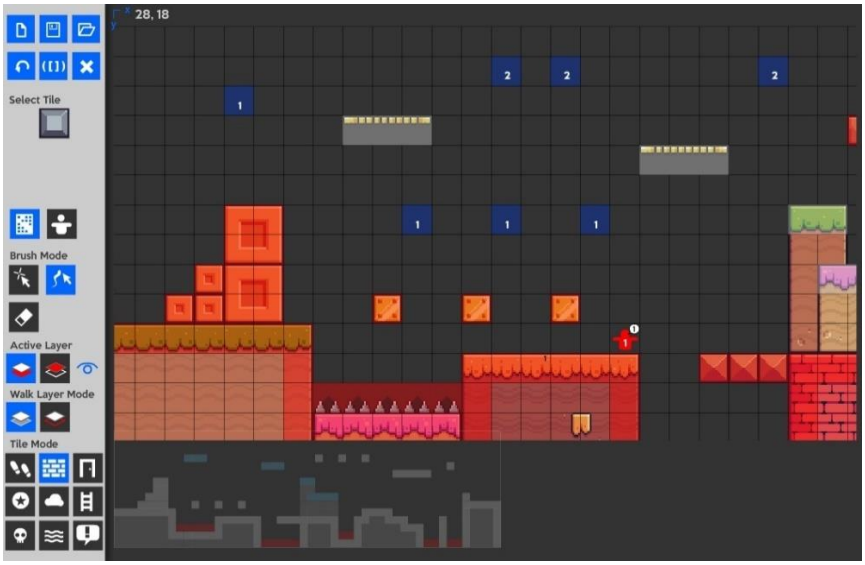
Gambar 89. Penambahan tujuan di akhir level

8.9 Menambahkan Beberapa Level

Pada **tutorial-2** memiliki 1 buah level, dalam dunia pengembangan *game* hal tersebut digunakan untuk melakukan tes atau ujicoba sistem (*engine test*). Ketika sistem sudah bekerja dengan cukup baik dalam artian fitur-fitur dasar sudah bekerja, maka tahapan selanjutnya adalah mengembangkan *game* dengan beberapa level yang lebih kompleks, lebih menarik dan memiliki kurva pengalaman yang membuat pemain cenderung ingin mengulang-ulang permainan.

Untuk menambahkan level pada tutorial-2 terdapat beberapa prosedur yang harus dilakukan, yaitu :

1. Membuat *database* peta dengan ID yang berbeda. Misal *file* **map_2.js** memiliki id peta 2 (anda dapat membuka *file* peta *array* dan memastikan nama variabelnya adalah `var map_2 = [ … ]`) . *File* peta tersebut harus diletakkan ke dalam *folder* **js**.



Gambar 90. Mengembangkan arena permainan level 2

2. Mengedit *file* **index.html** dan menambahkan kode untuk membuka *file* peta *array*. Misal untuk level 2, diperlukan tambahan kode untuk membuka *file* **map_2.js** sebagai berikut :

```
17. </body>
18. <script type="text/javascript" src="js/map_1.js"></script>
19. <script type="text/javascript" src="js/map_2.js"></script>
20. <script type="text/javascript" src="js/game.js"></script>
21. </html>
```

3. Menambahkan grafis untuk item dan musuh tambahan dan melakukan identifikasi pada variabel `gambar`, sekaligus melakukan identifikasi data animasi pada fungsi `setAwal`.
4. Melakukan sedikit perubahan pada fungsi `setAwal` agar dapat membuka *file* peta *array* secara dinamis. Pada baris :

   ```
   setPlatform(map_1, dataGambar.tileset, 32, game.hero);
   ```

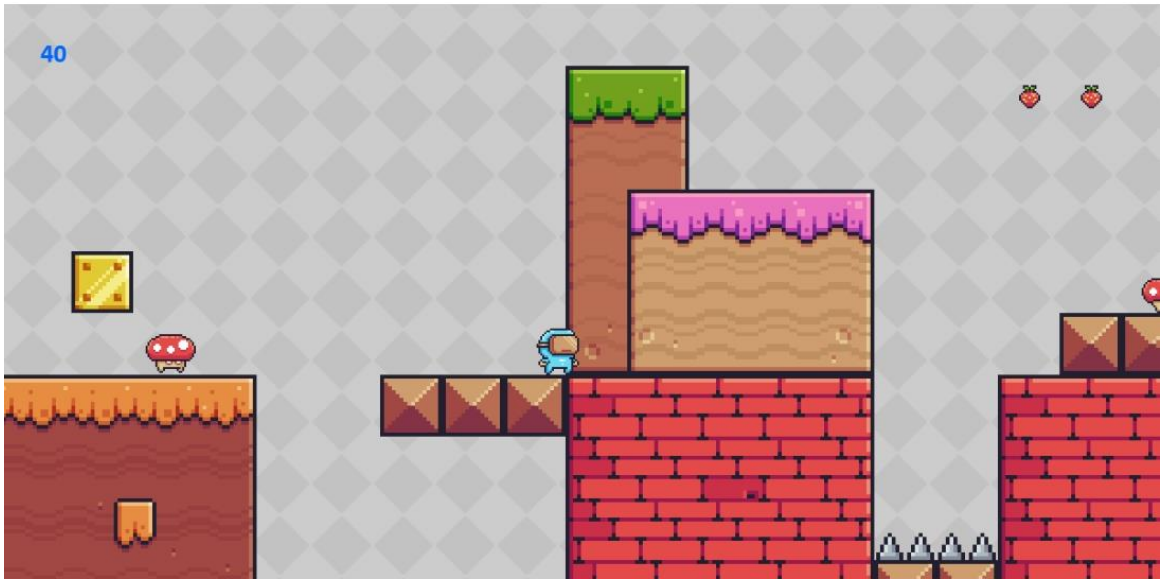
   diubah menjadi :

   ```
   setPlatform(this["map_"+game.level], dataGambar.tileset,
                32, game.hero);
   ```

5. Menambah variabel `game.level` ketika pemain berhasil menyentuh *trigger*. Hal ini dapat dilakukan pada fungsi `cekItem` sebagai berikut :

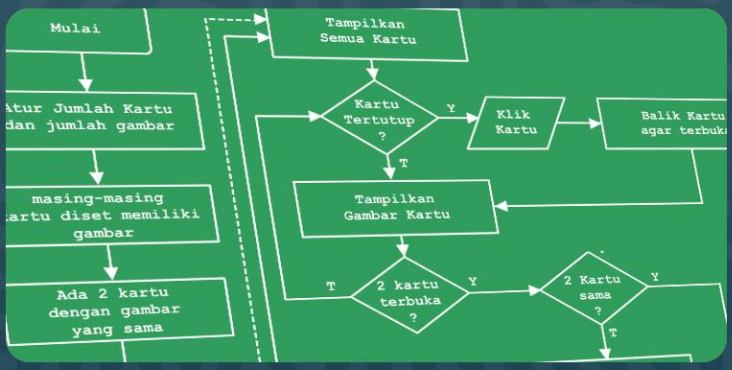
```
99. function cekItem(){
100.            . . .

107.            if (game.triggerID == 1){
108.                  game.triggerID = 0;
109.                  game.aktif = false;
110.                  game.level++;
111.                  setTimeout(ulangiPermainan, 2000);
112.            }
113.      }
```

6. Menambahkan suara dan *GUI* (antar muka). Penambahan suara dan GUI dapat mengadaptasi dari **tutorial-1**.



Gambar 91. Hasil penambahan level.

[halaman ini sengaja dikosongkan]



Flowchart

Bagi beberapa programmer, flowchart dapat dijadikan panduan untuk mengoptimalkan proses

BAB 9

GAME MENCOCOKAN GAMBAR

9.1 Konsep Dasar *Game* Mencocokan Gambar

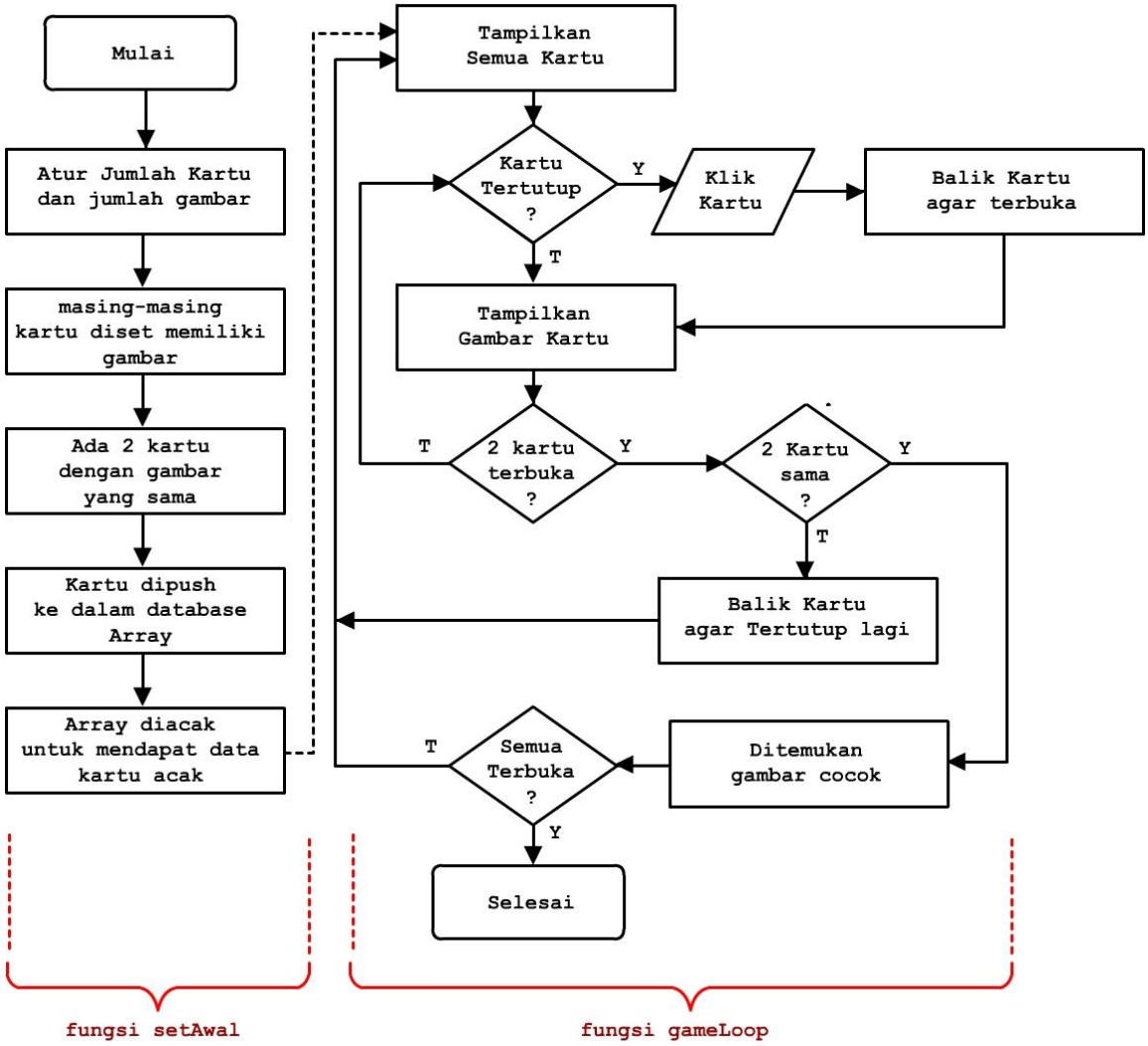
Pada bab ini akan dibuat **tutorial-3** yaitu *game* mencocokan gambar. *Game* mencocokan gambar merupakan *game* bertipe *puzzle* yang menampilkan beberapa kartu bergambar dan pemain harus mencari kartu dengan gambar yang sama. Melalui konsep *game* yang sederhana tersebut akan dipelajari teknik perulangan (*for loop*) dan teknik menggendalikan objek melalui *database array*.

Gambar 92. Tampilan *game* mencocokan gambar

Pada umumnya diperlukan proses *breakdown* tahapan-tahapan penting yang diperlukan dalam proses pengembangan *game*. Pada *game* mencocokan gambar misalnya, terdapat beberapa tahapan yang harus dilakukan sampai sistem permainan terbentuk. Dalam melakukan proses *breakdown* terdapat 2 tipe *programmer* atau pengembang *game*, yaitu tipe *programmer* yang menggunakan *flowchart*, merencanakan langkah-langkah sejak awal dan secara berurutan. Tipe kedua adalah *programmer* yang menyusun program secara dinamis, menambahkan kode secara bertahap namun tidak terikat urutan-urutan tertentu.

Pada tutorial pertama dan kedua, *game* dibuat dengan cara yang dinamis. Kode ditambahkan secara bertahap, variabel ditambahkan terus seiring kebutuhan, gambar ditambahkan pada saat dibutuhkan, dan kode akan diganti atau ditambah secara terus menerus sampai mendapatkan hasil yang diinginkan. Sementara, pada **tutorial-3** nantinya akan digunakan

flowchart untuk membantu pemrograman. *Flowchart* merupakan penggambaran secara grafik dari langkah-langkah dan urut-urutan prosedur dari suatu program. *Flowchart* dapat membantu programmer untuk memecahkan masalah kedalam segmen-segmen yang lebih kecil, menganalisis alternatif-alternatif yang dapat dilakukan, dan menyusun variabel-variabel yang dibutuhkan dalam pemrograman sejak awal (Takatalo dkk, 2015). Perhatikan *flowchart* berikut :



Gambar 93. *Flowchart game* mencocokan gambar

Dalam *flowchart* di atas, dapat ditentukan variabel dan langkah-langkah yang dibutuhkan untuk menyusun *game* mencocokan gambar. Dimulai dari pengaturan awal, yaitu mengatur jumlah kartu, jumlah gambar yang dipakai, serta langkah untuk mencocokan gambar sampai seluruh kartu terbuka. Dari *flowchart* tersebut diperoleh gambaran variabel dan aset yang dibutuhkan untuk menyusun *game* mencocokan gambar yang nantinya dilakukan pada fungsi

`setAwal`. Selanjutnya, kita juga dapat merencanakan apa yang harus dilakukan pada fungsi `gameLoop` yaitu menampilkan seluruh kartu dalam kondisi tertutup, menunggu input dari pemain untuk membuka kartu, menunggu hingga dua kartu terbuka dan menutup kembali apabila kartu tidak sama, atau melanjutkan membuka kartu lainnya jika kedua kartu tersebut sama.

9.2 Persiapan Aset Grafis

Untuk membuat *game* mencocokan gambar diperlukan sebuah gambar (*spritesheet*) yang berisi beberapa gambar yang akan dimunculkan di kartu. Pada tutorial ini digunakan gambar beberapa karakter Pokemon (*copyright* Nintendo). Satu *spritesheet* berisi 19 gambar berukuran 128 x 128 *pixel* dan satu buah *card deck* (bagian belakang kartu untuk menampilkan kartu dalam kondisi tertutup). Gambar disimpan dalam format *file* PNG dan nantinya diletakkan ke dalam *folder* **assets**.

Gambar 94. *Spritesheet* **cards.png** untuk *game* mencocokan gambar

Gambar selanjutnya yang dibutuhkan adalah indikator untuk menunjukkan apabila permainan berhasil dimenangkan dengan cara membuka semua gambar. Gambar berupa sebuah tulisan “menang !!” dan disimpan dengan nama **menang.png**

Gambar 95. Sprite **menang.png** sebagai indikator ketika *game* dimenangkan

Gambar terakhir adalah gambar halaman judul yaitu **cover.jpg** yang nantinya digunakan untuk menggantikan gambar cover yang sudah ada di *folder* **assets** di dalam *folder* ***template project*** .

Gambar 96. Cover *game* mencocokan gambar

9.3 Pemrograman *Game* Mencocokan Gambar

Untuk membuat *game* mencocokan gambar, ikutilah langkah-;angkah berikut :

1. Copy *folder* ***template***, *paste* dan ubah nama *folder* menjadi **tutorial-3**.
2. Letakkan aset gambar (*file* **cards.png, menang.png** dan **cover.jpg**) ke dalam *folder* **assets**.
3. Buka *file* **game.js**, kemudian ubah kode menjadi sebagai berikut :

```
1.  setGame("1200x600");
2.  game.folder  = "assets";
3.  //file gambar yang dipakai dalam game
4.  var gambar = {
5.          logo:"logo.png",
6.          startBtn:"tombolStart.png",
7.          cover:"cover.jpg",
8.          playBtn:"btn-play.png",
9.          maxBtn:"maxBtn.png",
10.         minBtn:"minBtn.png",
11.         cards:"cards.png",
12.         menang:"menang.png"
13. }
14. //file suara yang dipakai dalam game
15. var suara = {
16. }
17.
18. //load gambar dan suara lalu jalankan startScreen
19. loading(gambar, suara, startScreen);
20.
21. function startScreen(){
22.         hapusLayar("#67d2d6");
23.         tampilkanGambar(dataGambar.logo, 600, 250);
```

```
24.         var startBtn = tombol(dataGambar.startBtn, 600, 350);
25.         if (tekan(startBtn)){
26.                 jalankan(halamanCover);
27.         }
28. }
29. function halamanCover(){
30.         hapusLayar("#67d2d6");
31.         gambarFull(dataGambar.cover);
32.         var playBtn = tombol(dataGambar.playBtn, 1115, 515);
33.         if (tekan(playBtn)){
34.                 if (game.aktif) {
35.                         //mulai game dengan menambahkan transisi
36.                         game.status = "mulai";
37.                         game.level = 1;
38.                         game.score = 0;
39.                         game.warnaTransisi = "#67d2d6";
40.                         transisi("out", setAwal);
41.                 }
42.         }
43.         resizeBtn(1150,50);
44.         efekTransisi();
45. }
46.
47. function setAwal(){
48.         //atur jumlah gambar maksimal yang bisa muncul
49.         game.kartuMax = 10;
50.         //atur jumlah kartu di layar (misal 20 kartu)
51.         game.kartuX = 5;
52.         game.kartuY = 4;
53.         //seting kartu
54.         game.kartuBenar = 0;
55.         game.jumlahKartu = game.kartuX*game.kartuY;
56.         game.kartuTerbuka = 0;
57.         game.kartuDB = [];
58.         game.jenisKartu = 0;
59.         //mengacak 19 gambar yang ada, gambar ke 20 untuk kartu tertutup
60.         game.kartuAcak = acakAngka("1-19");
61.         for (var i = 1; i <= game.jumlahKartu/2; i++){
62.                 game.jenisKartu++;
63.                 if (game.jenisKartu > game.kartuMax) game.jenisKartu = 1;
64.                 //buat objek baru untuk menyimpan data kartu
65.                 game.card = setSprite(dataGambar.cards, 128, 128);
66.                 game.card.jenisKartu = game.kartuAcak[game.jenisKartu];
67.         //status kartu 0 = tertutup 1=terbuka 2=flip buka 3=flip tutup
68.                 game.card.stat = 0;
```

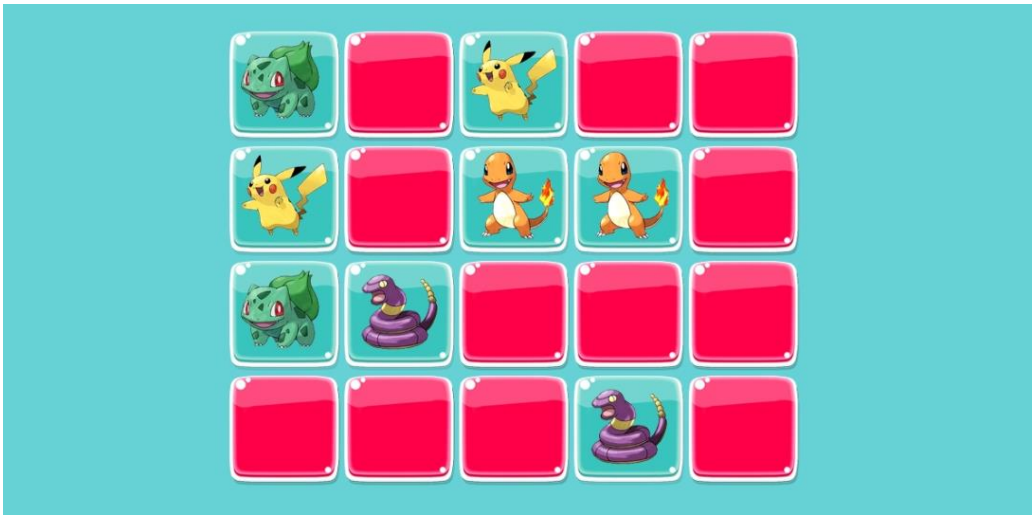
```
69.                 game.kartuDB.push(game.card);
70.                 //buat kartu kedua dengan data yang sama,
71.                 //karena pasti ada 2 kartu yang sama
72.                 game.card2 = setSprite(dataGambar.cards, 128, 128);
73.                 game.card2.jenisKartu = game.kartuAcak[game.jenisKartu];
74.                 game.card2.stat = 0;
75.                 game.kartuDB.push(game.card2);
76.         }
77.         //acak kartu
78.         game.kartuDB = acakArray(game.kartuDB);
79.         //setelah semua pengaturan selesai jalankan gameLoop
80.         game.menang = false;
81.         game.status = "main";
82.         jalankan(gameLoop);
83.         transisi("in");
84. }
85.
86. function gameLoop(){
87.         hapusLayar();
88.         tampilkanKartu();
89.         cekKartu();
90.         cekMenang();
91.         efekTransisi();
92. }
93.
94. function tampilkanKartu(){
95.         //tampilkan kartu via for 2 tingkat
96.         for (var i=0;i < game.kartuX;i++){
97.                 for (var j=0;j < game.kartuY;j++){
98.                         //pengaturan gambar kartu
99.                         var id = i*game.kartuY+j;
100.                                 game.card = game.kartuDB[id];
101.                                 //tampilkan kartu sesuai dengan kondisi stat
102.                                 if (game.card.stat == 0) {
103.                                         game.card.frame = 20;
104.                                 }else if (game.card.stat == 1){
105.                                         game.card.frame = game.card.jenisKartu;
106.                                 }
107.                         game.card.id = id;
108.                         //berikan jarak 5 pixel antar kartu
109.                         var sx = (game.lebar - game.kartuX*
                            (game.card.lebar+5))/2 + game.card.lebar/2;
110.                         var sy = (game.tinggi - game.kartuY*
                            (game.card.tinggi+5))/2 + game.card.tinggi/2;
111.                         game.card.x = sx+i*(game.card.lebar+5);
```

```
112.                    game.card.y = sy+j*(game.card.tinggi+5);
113.                    //tampilkan kartu sesuai dengan kondisi
114.                    if (game.card.stat < 2) sprite(game.card);
115.                    if (game.card.stat == 2) putar(game.card);
116.                    if (game.card.stat == 3) putarBalik(game.card);
117.                    //deteksi input pemain jika mouse di klik
118.                    if (game.mouseDitekan && game.kartuTerbuka<2){
119.                           if (cekHit(game.mouse, game.card) &&
                                   game.card.stat == 0 ){
120.                              game.card.stat = 2;
121.                              game.kartuTerbuka++;
122.                              //menyimpan data kartu yang terbuka
123.                              if (game.kartuTerbuka == 1)
                                 game.kartu1 = game.card;
124.                              if (game.kartuTerbuka == 2)
                                 game.kartu2 = game.card;
125.                              //mengatur kondisi kartu di awal dalam
126.                              game.card.buka = 1;
127.                              game.card.skalaX = 1;
128.                              //waktu tunggu jika kartu tidak cocok
129.                              game.waktuTutup = 100;
130.                              }
131.                           }
132.                    }
133.             }
134.      }
135.
136.      function putar(card){
137.             card.buka++;
138.             //efek balik kartu
139.             if (card.buka < 10){
140.                    //kartu tertutup mulai dibalik
141.                    card.frame = 20;
142.                    card.skalaX-=0.1;
143.                    if (card.skalaX <= 0) card.buka = 10;
144.             }else if (card.buka >= 10 && card.buka < 20){
145.                    //kartu mulai terbalik dan menampilkan gambar
146.                    card.frame = card.jenisKartu;
147.                    card.skalaX+=0.1;
148.                    if (card.skalaX > 1) card.buka = 20;
149.             }else if (card.buka >= 20){
150.                    //kartu terbuka sempurna
151.                    card.skalaX = 1;
152.                    card.frame = card.jenisKartu;
153.                    card.stat = -1;
```

```
154.            }
155.            //tampilkan kartu
156.            sprite(card);
157.      }
158.
159.      function putarBalik(card){
160.            card.buka++;
161.            //efek balik kartu
162.            if (card.buka < 10){
163.                  //dimulai dari kartu bergambar
164.                  card.frame = card.jenisKartu;
165.                  card.skalaX-=0.1;
166.                  if (card.skalaX <= 0) card.buka = 10;
167.            }else if (card.buka >= 10 && card.buka < 20){
168.                  //ketika kartu bergambar dibalik,
169.                  //maka dimunculkan dek kartu
170.                  card.frame = 20;
171.                  card.skalaX+=0.1;
172.                  if (card.skalaX > 1) card.buka = 20;
173.            }else if (card.buka >= 20){
174.                  card.skalaX = 1;
175.                  card.frame = 20;
176.                  //kartu tertutup
177.                  card.stat = 0;
178.                  game.kartuTerbuka--;
179.                  if (game.kartuTerbuka < 0) game.kartuTerbuka = 0;
180.            }
181.            sprite(card);
182.      }
183.
184.      function cekKartu(){
185.            //cek apakah ada 2 kartu yang terbuka?
186.            if (game.kartuTerbuka == 2){
187.                  //dan pastikan 2 kartu sudah terbuka sempurna
188.                  if (game.kartu1.stat == -1 &&
189.                      game.kartu2.stat == -1){
190.                  if (game.kartu1.jenisKartu ==
                          game.kartu2.jenisKartu){
191.                        //pilihan kartu benar
192.                        game.kartuBenar+=2;
193.                        game.kartuTerbuka = 0;
194.                        //menang apabila semua kartu terbuka
195.                        if (game.kartuBenar >= game.jumlahKartu){
196.                              //permainan selesai dan menang
197.                              game.menang = true;
```

```
198.                         }
199.                         }else{
200.                         //kartu berbeda,
                               tunggu sesaat untuk menutup kembali
201.                             game.waktuTutup--;
202.                             //klik, maka akan segera menutup
203.                             if (game.mouseDitekan){
204.                                 game.mouseDitekan = false;
205.                                 game.waktuTutup = 5;
206.                             }
207.                             if (game.waktuTutup == 0){
208.                                 game.kartu1.stat = 3;
209.                                 game.kartu2.stat = 3;
210.                                 game.kartu1.buka = 1;
211.                                 game.kartu2.buka = 1;
212.                             }
213.                         }
214.                     }
215.             }
216.     }
217.
218.     function cekMenang(){
219.             if (game.menang){
220.                     kotak(0,0, game.lebar, game.tinggi, 1, "#000",
                        "#000", 50);
221.                     //gambar "menang" akan masuk ke tengah layar
222.                     efekMasuk("menang", dataGambar.menang, game.lebar/2,
                                game.tinggi/2-50, "kiri");
223.                     teks("Klik untuk mengulang", game.lebar/2,
                            game.tinggi/2,
224.                            "Calibri-bold-15pt-tengah-merah|putih-kedip");
225.                     //jika ditekan permainan akan diulang
226.                     if (game.mouseDitekan && game.status == "main"){
227.                             game.status = "mulai";
228.                             game.mouseDitekan = false;
229.                             transisi("out", setAwal);
230.                     }
231.             }
232.     }
```

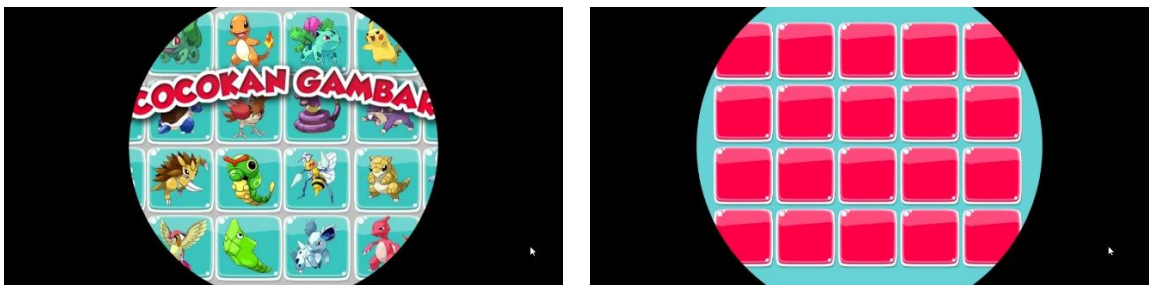
4. Simpan *file* **game.js**, dan jalankan *file* **index.html**. Maka akan diperoleh sebuah *game* mencocokkan gambar.



Gambar 97. Hasil *game* mencocokan gambar

Penjelasan Program

Pada dasarnya kode di atas memiliki komentar kode (`//`) yang merupakan penjelasan dari langkah kode yang dimaksud. Sedikit perbedaan dengan tutorial sebelumnya, pada fungsi `halamanCover` pada saat tombol `play` ditekan akan ditambahkan `transisi`. Transisi akan menghasilkan sebuah perpindahan halaman dengan efek transisi *wipe*, sehingga tampilan menjadi lebih menarik. Kode transisi memiliki parameter "`in`" dan "`out`", dan dapat menjalankan fungsi tertentu di akhir efek. Sebagai contoh, kode `transisi("out", setAwal);` akan memberikan efek *wipe out* (lingkaran menutup layar) dan kemudian menjalankan fungsi `setAwal`.



Gambar 98. Efek *wipe out* dan *wipe in*

Agar efek transisi tersebut dapat dimunculkan di *canvas*, maka pada baris akhir pada fungsi yang dijalankan, ditambahkan kode `efekTransisi();`

Pada fungsi `setAwal` dilakukan pengaturan sebagaimana urutan dalam *flowchart*. Kartu diatur memiliki jumlah baris dan kolom, kemudian masing-masing kartu ditambahkan gambar. Operasi `for` akan menduplikasi kartu masing-masing sebanyak 2 kali karena selalu

ada 2 kartu dengan gambar yang sama. Data kartu selanjutnya disimpan dalam variabel *array* dan diacak menggunakan kode `game.kartuDB = acakArray(game.kartuDB);`

Pada saat fungsi `gameLoop` dijalankan, maka terdapat 4 langkah sesuai dengan *flowchart*, yaitu menampilkan kartu dan menunggu input pemain (fungsi `tampilkanKartu`), mengecek kartu yang sama (fungsi `cekKartu`) dan mengecek kondisi ketika semua kartu terbuka (fungsi `cekMenang`). Ke 4 langkah dalam 3 fungsi tersebut akan dijalankan secara terus menerus sampai *game* berhasil dimenangkan.

Ketika menang, ditambahkan sebuah efek gambar "menang" yang bergerak masuk ke tengah layar melalui fungsi `efekMasuk( nama objek, dataGambar, tujuanX, tujuanY, asal);` dan diberikan efek `teks` berkedip. Sesuai dengan *flowchart*, ketika permainan selesai dan di klik maka permainan akan memulai dengan acakan kartu baru dan permainan akan diulang kembali.

Pada tahapan ini pembuatan *game* mencocokan gambar dapat ditambahkan *timer* (waktu permainan), ditambahkan level dengan cara menaikkan nilai beberapa variabel:

`game.kartuMax = 10;`
untuk mengatur jumlah gambar yang ditampilkan di dalam kartu, sehingga semakin banyak gambar permainan akan menjadi semakin sulit

`game.kartuX = 5;` dan `game.kartuY = 4;`
untuk mengatur kolom dan baris kartu. Apabila level naik, maka jumlah baris dan kolom bisa ditambah untuk meningkatkan level kesulitan dalam game.

Penambahan suara, *GUI* dan score juga akan menambah daya tarik *game* untuk dimainkan secara berulang.

Actionscript & Javascript

Kedua bahasa tersebut memiliki banyak kesamaan sehingga memporting tutorial AS ke JS pada dasarnya tidak terlalu rumit karena struktur dan syntax yang relatif sama

BAB 10
KUIS

10.1 Mengadaptasi Tutorial Program Lain (*Porting*)

Sebelum HTML5 *canvas* dan *javascript* digunakan, telah banyak bahasa pemrograman lainnya yang dapat digunakan untuk membuat aplikasi berbasis web. Tutorial yang tersedia juga cukup banyak, oleh karena itu dalam sebuah pengembangan aplikasi atau *game* dapat dilakukan proses *porting*. *Porting* merupakan proses menerjemahkan dari satu bahasa pemrograman atau protokol ke bahasa pemrograman yang lain. Bahasa "diterjemahkan", dan data akan "dikonversi". *Porting* menyiratkan perubahan instruksi namun bukan struktur data, sehingga prosedur atau fungsi yang ada dapat diturunkan ke dalam bahasa baru.

Sebagai contoh, pada bab ini akan dijelaskan tentang proses pembuatan kuis yang diadaptasi dari tutorial Adobe Animate/Flash. Tutorial yang akan di*porting* adalah tutorial membuat kuis pada link https://www.wandah.org/membuat-*game*-kuis-dengan-flash. Pada tutorial tersebut soal dalam format *array* akan diacak kemudian ditampilkan ke layar dalam bentuk soal pilihan ganda.

Gambar 99. *Game* kuis flash

Adobe Animate/Adobe Flash merupakan program *authoring tools* yang artinya pada aplikasi tersebut seluruh aset visual dan kode dapat diatur sedemikian rupa pada editor khusus (Spuy, 2012). Sementara, pengembangan *game* di buku ini hanya menggunakan teks/*script editor*

sehingga manajemen aset gambar maupun suara harus diatur secara manual. Selebihnya selain manajemen aset, penggunaan kode *Actionscript* pada Adobe Animate/Flash dan *javascript* tidak terlalu berbeda.

Berikut perbedaan antara pengembangan *game* kuis dengan Adobe Animate/Flash dibandingkan dengan pengembangan dengan menggunakan HTML5 *canvas javascript*:

Tabel 1. Perbandingan antara Flash dan *Javascript* dalam pengembangan *game* kuis

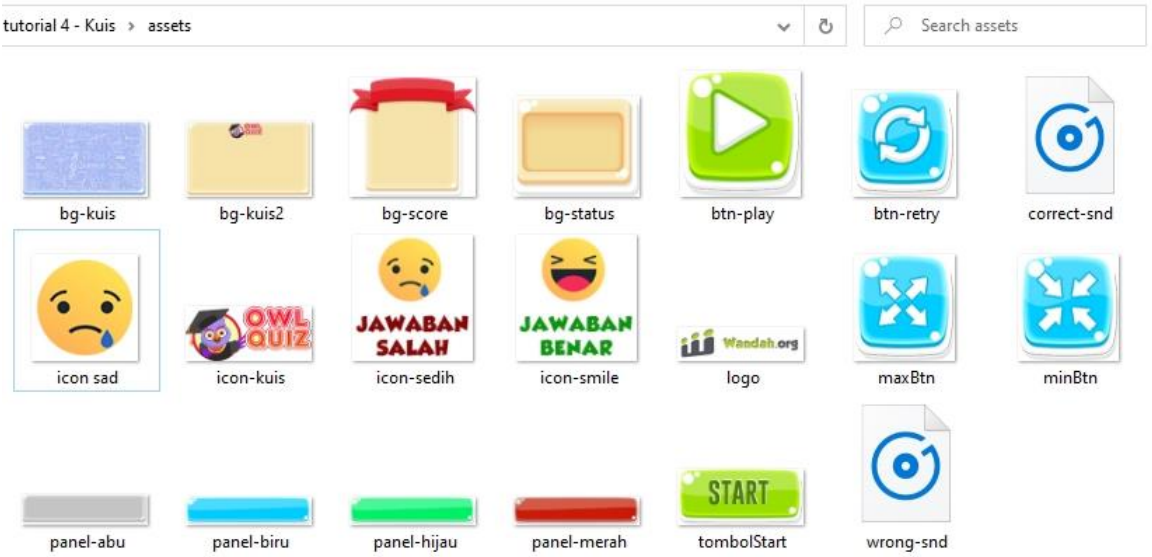
Aspek	**Adobe Animate/Flash**	***Javascript***
Aset Visual	Dibuat langsung pada aplikasi Atau di*import* dari luar	Dibuka sebagai elemen DOM HTML (di*import*)
Menata tampilan visual	Drag objek ke layar atau menggunakan kode	Menggunakan kode
Format Soal Kuis	*Array*	Sama dan *file* dari tutorial bisa digunakan secara langsung
Mengacak Soal	Fungsi acak soal	Kode : `acakArray(soal)`
Menampilkan soal	*Dynamic Text* dan kode	Kode : `teksHTML`
Menampilkan jawaban	Movieclip dan kode	Kode : `tampilkanGambar` dan `tombol`
Cek Jawaban	Fungsi cek jawaban	Sama
Nilai akhir	Movieclip dan kode	Kode : `tampilkanGambar` dan `teks`

Dari tabel perbandingan di atas, untuk mem*porting game* kuis dari Adobe Animate/Flash menuju ke HTML5 *canvas* pada dasarnya hanya memiliki perbedaan pada cara menampilkan gambar ke layar. Animate/Flash merupakan *authoring tools* sehingga untuk mengatur objek dapat dilakukan dengan teknik *drag and drop*, sementara *javascript* yang hanya menggunakan teks editor menggunakan peletakan gambar harus dengan kode. Selebihnya fungsi dan teknik pemrograman kurang lebih sama. Pada titik ini, kelebihan dari HTML5 *canvas*, selain dapat dijalankan di *browser* tanpa plugin aplikasi HTML 5 *canvas* dapat dikembangkan secara gratis.

10.2 Asset untuk *Game* Kuis

Untuk memulai membuat *game* kuis, terlebih dahulu kita menduplikasi *folder* ***template*** dan mengubah namanya menjadi *folder* **tutorial-4**. Selanjutnya, aset yang digunakan untuk *game* kuis akan mengadaptasi dari *game* yang akan di*porting*, yaitu *game* kuis berbasis Adobe Animate/Flash. Gambar yang digunakan adalah gambar "mentah" yang sebelum di*import* ke Adobe Flash. Gambar tersebut meliputi gambar latar, gambar untuk tombol, dan gambar

untuk *pop up* status jawaban pemain. Seluruh gambar tersebut diletakkan ke dalam *folder* **assets**, berikut 2 *file* suara untuk suara jawaban benar dan suara jawaban salah. Perhatikan isi dari *folder* **assets** setelah penambahan aset dari *game* kuis Flash sebagai berikut :



Gambar 100. Isi dari *folder* **assets** tutorial-4.

10.3 *Database* Soal

Pada tutorial kuis Adobe Flash/Animate digunakan variabel soal bertipe *array* sebagai berikut:

```
1. var soal:Array = [["Siapakah penemu mesin uap ?", "James Watt",
   "Davinci", "Issac Newton", "James bond"],
2.                   ["Akar dari 676 adalah ?", "26", "24", "16", "34"],
3.                   ["Bagian tumbuhan yang berperan penting dalam
   fotosintesis adalah ?", "Klorofil", "Kambium", "Epidermis", "Kromatin"],
4.                   ["Penulis trilogi Lord of The Ring adalah?", "JRR
   Tolkien", "JK Rowling", "J Cameron", "J Thomas"],
5.                   ["Jumlah seluruh sudut segitiga siku-siku adalah ?",
   "180", "270", "90", "360"],
6.                   ["Provinsi termuda di Indonesia adalah ? ", "Banten",
   "Bangka belitung", "Batam", "Gorontalo"], . . .];
```

Struktur variabel soal bertipe array tersebut adalah meletakkan soal pada data *array* ke 0 (perhitungan *array* dimulai dari angka 0), meletakkan jawaban benar pada data *array* ke 1, dan meletakkan 3 jawaban lainnya pada data *array* ke 2-4. Sehingga apabila kode

```
trace(soal[0][0]);
```

akan menghasilkan urutan soal ke 0 dan data nomor 0 , yaitu "`Siapakah penemu mesin uap?`".

Porting variabel tersebut ke *javascript* relatif mudah, karena memiliki struktur yang sama, hanya saja bagian deklarasi variabel (`:Array`) perlu di hapus, karena pada *javascript* variabel tidak perlu didefinisikan dengan detail. Sehingga pada *porting javascript* kode tersebut menjadi :

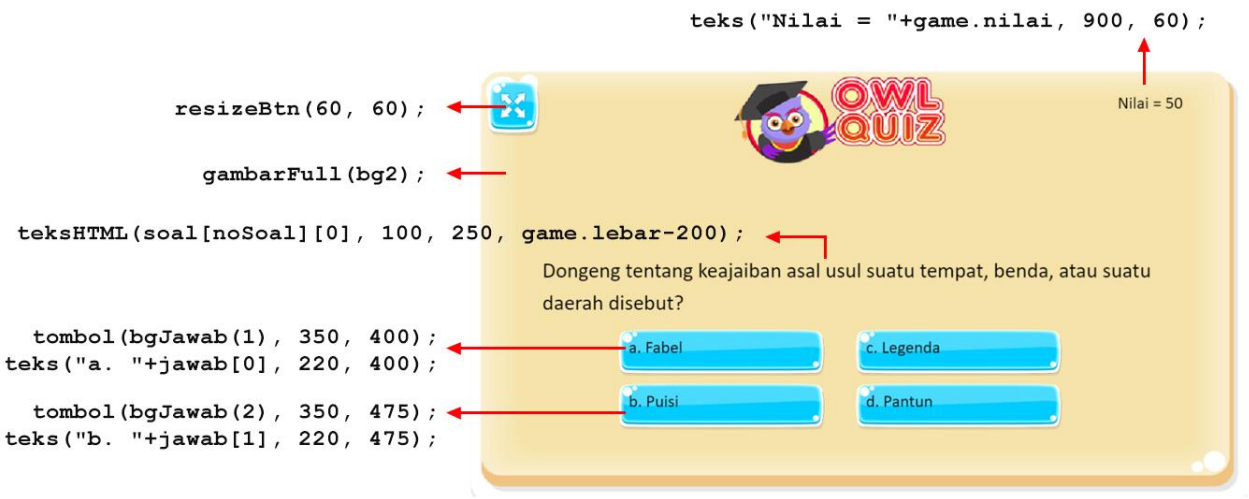
```
1. var soal = [["Siapakah penemu mesin uap ?", "James Watt", "Davinci",
   "Issac Newton", "James bond"],
2.                     ["Akar dari 676 adalah ?", "26", "24", "16", "34"],
3.                     ["Bagian tumbuhan yang berperan penting dalam
   fotosintesis adalah ?", "Klorofil", "Kambium", "Epidermis", "Kromatin"],
4.                     ["Penulis trilogi Lord of The Ring adalah?", "JRR
   Tolkien", "JK Rowling", "J Cameron", "J Thomas"],
5.                     ["Jumlah seluruh sudut segitiga siku-siku adalah ?",
   "180", "270", "90", "360"],
6.                     ["Provinsi termuda di Indonesia adalah ? ", "Banten",
   "Bangka belitung", "Batam", "Gorontalo"],. . . ];
```

Untuk mempermudah pembuatan aplikasi, *file* soal dapat disimpan secara terpisah dengan nama **soal.js** dan diletakkan ke dalam *folder* **js**. Untuk mengakses *file* tersebut, pada *file* **index.html** perlu ditambahkan satu baris kode sebagai berikut :

```
22. </body>
23. <script type="text/javascript" src="js/soal.js"></script>
24. <script type="text/javascript" src="js/game.js"></script>
25. </html>
```

10.4 Pemrograman Antarmuka (*interface*)

Pada tabel 1 dijelaskan bahwa perbedaan utama pada *game* kuis berbasis Adobe Flash/Animate dan HTML 5 *canvas* adalah pada cara menampilkan aset visual. Pada HTML5 *canvas*, aset visual harus diletakkan dengan menggunakan kode. Secara spesifik apabila tampilan kuis tersebut kita *breakdown* ke dalam kode *javascript* maka akan didapatkan struktur kode seperti pada gambar berikut :



Gambar 101. Stuktur kode *interface game* kuis

Untuk membentuk antarmuka *game* kuis dengan menggunakan kode, kita perlu menuliskan urutan kode berdasarkan posisi *layer* (lapisan). Semakin bawah *layer* (posisi gambar semakin belakang), maka kode dituliskan terlebih dahulu sehingga grafis akan di*render* ke *canvas* terlebih dahulu. Sehingga kode untuk membentuk antarmuka di atas nantinya akan diletakkan pada fungsi `gameLoop` sebagai berikut :

```
1.  function gameLoop(){
2.          hapusLayar();
3.          gambarFull(dataGambar.bg2);
4.          //teks soal
5.          game.font = "Calibri-normal-30pt-left-hitam";
6.          teksHTML(game.tempSoal[game.nomorSoal][0], 100, 250,
                    game.lebar-200);
7.          //tampilkan background dan jawaban
8.          acakJawaban();
9.          game.font = "Calibri-normal-18pt-left-hitam";
10.         game.jawab1 = tombol(bgJawab(1), 350, 400);
11.         teks("a. "+game.tempJawaban[0], 220, 400);
12.         game.jawab2 = tombol(bgJawab(2), 350, 475);
13.         teks("b. "+game.tempJawaban[1], 220, 475);
14.         game.jawab3 = tombol(bgJawab(3), 680, 400);
15.         teks("c. "+game.tempJawaban[2], 550, 400);
16.         game.jawab4 = tombol(bgJawab(4), 680, 475);
17.         teks("d. "+game.tempJawaban[3], 550, 475);
18.         //tampilkan nilai
19.         teks("Nilai = "+game.nilai, 900, 60);
20.         . . .
```

```
21.
22.         resizeBtn(60,60);
23. }
```

Pada awal langkah dibuat sebuah gambar penuh sebagai latar dengan menggunakan kode `gambarFull(dataGambar);` selanjutnya ditampilkan soal dengan menggunakan kode

```
teksHTML (teks, posisiX, posisiY, lebar area teks);
```

Berbeda dengan kode `teks`, kode `teksHTML` memungkinkan untuk menampilkan teks dalam format beberapa baris (*multiline*). Pada aplikasi Adobe Animate/Flash langkah ini dapat dilakukan dengan membuat *dynamic text*, pada *javascript* cukup dengan menentukan posisi x, posisi y serta lebar area dari teks yang akan ditampilkan. Sesuai nama fungsinya, kode `teksHTML` tersebut juga mendukung tag HTML seperti **<b>**, **
, **<i>, **<sub>** dan **<sup>**, yang dapat dimanfaatkan untuk menampilkan teks yang memiliki format lebih kompleks. Misal ingin menuliskan soal "`Berapakah nilai dari` 5^2+4^3 `= ?`", maka dapat ditulis dengan kode

```
teksHTML("Berapakah nilai dari 5<sup>2</sup> + 4<sup>3</sup>= ?",
100, 250, 600);
```

Untuk menampilkan jawaban diperlukan trik khusus. Apabila pada aplikasi Adobe Animate/Flash dapat digunakan *movieclip* untuk menampilkan tombol jawaban dengan latar belakang warna yang berbeda-beda, maka pada *javascript* gambar tombol diatur menggunakan fungsi `bgJawab(noJawaban)`, agar latar tombol nantinya dapat diubah sesuai kebutuhan. Apabila anda lihat pada *folder* **assets**, terdapat 3 warna yang berbeda pada latar belakang jawaban, yaitu warna biru (soal belum dijawab) , hijau (jawaban benar), dan merah (jawaban salah), sehingga pada baris :

```
10.         game.jawab1 = tombol(bgJawab(1), 350, 400);
11.         teks("a. "+game.tempJawaban[0], 220, 400);
```

akan ditampilkan sebuah latar gambar salah satu panel warna, kemudian di atasnya akan dituliskan opsi jawaban.

Untuk menampilkan nilai digunakan kode `teks` dan untuk menampilkan tombol *resize* digunakan kode `resizeBtn`.

10.5 Pemrograman *Game* Kuis

Setelah memahami teknik membangun antarmuka pada *game* kuis, maka secara keseluruhan kode pada *file* **game.js** adalah sebagai berikut :

```
1.  setGame("1080x600");
2.  game.folder  = "assets";
3.  //file gambar yang dipakai dalam game
4.  var gambar = {
5.          bg:"bg-kuis.jpg",
6.          bg2:"bg-kuis2.jpg",
7.          logo:"logo.png",
8.          maxBtn:"maxBtn.png",
9.          minBtn:"minBtn.png",
10.         startBtn:"tombolStart.png",
11.         icon:"icon-kuis.png",
12.         play:"btn-play.png",
13.         bgJawab1:"panel-biru.png",
14.         bgJawab2:"panel-hijau.png",
15.         bgJawab3:"panel-merah.png",
16.         bgStatus:"bg-status.png",
17.         iconBenar:"icon-smile.png",
18.         iconSalah:"icon-sedih.png",
19.         bgScore:"bg-score.png",
20.         ulang:"btn-retry.png"
21. }
22. //file suara yang dipakai dalam game
23. var suara = {
24.         benar:"correct-snd.mp3",
25.         salah:"wrong-snd.mp3"
26. }
27.
28. //load gambar dan suara lalu jalankan startScreen
29. loading(gambar, suara, startScreen);
30.
31. function startScreen(){
32.         hapusLayar("#67d2d6");
33.         var startBtn = tombol(dataGambar.startBtn, game.lebar/2,
                                  game.tinggi/2+70);
34.         tampilkanGambar(dataGambar.logo, game.lebar/2, game.tinggi/2-25);
35.         if (tekan(startBtn)){
36.                 jalankan(gameCover);
37.         }
38. }
39.
```

```
function gameCover(){
        hapusLayar();
        gambarFull(dataGambar.bg);
        tampilkanGambar(dataGambar.icon, game.lebar/2, game.tinggi/2-100);
        var playBtn = tombol(dataGambar.play, game.lebar/2,
                                game.tinggi/2+100);
        resizeBtn(60,60);
        if (tekan(playBtn)){
                setAwal();
        }
}

function setAwal(){
        acakSoal();
        jalankan(gameLoop);
}

function gameLoop(){
        hapusLayar();
        gambarFull(dataGambar.bg2);
        //teks soal
        game.font = "Calibri-normal-30pt-left-hitam";
        teksHTML(game.tempSoal[game.nomorSoal][0], 100, 250,
                  game.lebar-200);
        //tampilkan background dan jawaban
        acakJawaban();
        game.font = "Calibri-normal-18pt-left-hitam";
        game.jawab1 = tombol(bgJawab(1), 350, 400);
        teks("a. "+game.tempJawaban[0], 220, 400);
        game.jawab2 = tombol(bgJawab(2), 350, 475);
        teks("b. "+game.tempJawaban[1], 220, 475);
        game.jawab3 = tombol(bgJawab(3), 680, 400);
        teks("c. "+game.tempJawaban[2], 550, 400);
        game.jawab4 = tombol(bgJawab(4), 680, 475);
        teks("d. "+game.tempJawaban[3], 550, 475);
        //tampilkan nilai
        teks("Nilai = "+game.nilai, 900, 60);
        //cek ketika tombol ditekan
        if (game.stat == 1){
                if (tekan(game.jawab1)) cekJawaban(1);
                if (tekan(game.jawab2)) cekJawaban(2);
                if (tekan(game.jawab3)) cekJawaban(3);
                if (tekan(game.jawab4)) cekJawaban(4);
        }

```

```
83.       if (game.stat > 1)tampilkanHasil();
84.
85.       resizeBtn(60,60);
86. }
87.
88. function tampilkanHasil(){
89.       if (game.timer>30){
90.             var skl = game.timer*2;
91.             if (skl > 100) skl = 100;
92.             tampilkanGambar(dataGambar.bgStatus, game.lebar/2,
                              game.tinggi/2-80, "skala="+skl);
93.             //jawaban benar
94.             if (game.stat == 2){
95.                   tampilkanGambar(dataGambar.iconBenar, game.lebar/2,
                                    game.tinggi/2-90, "skala="+skl);
96.             }
97.             //jawaban salah
98.             if (game.stat == 3){
99.                   tampilkanGambar(dataGambar.iconSalah, game.lebar/2,
                                    game.tinggi/2-90, "skala="+skl);
100.                  }
101.                  //restart soal
102.                  if (game.timer>120){
103.                        game.stat = 1;
104.                        game.nomorSoal++;
105.                        game.acak = 0;
106.                        //soal habis
107.                        if (game.nomorSoal>=game.soalMaks){
108.                              jalankan(halamanNilai);
109.                        }
110.                  }
111.            }
112.      }
113.
114.      function bgJawab(num){
115.            if (game.stat == 1){
116.                  return dataGambar.bgJawab1;
117.            }else{
118.                  if(game.tempJawaban[num-1] ==
                        game.tempSoal[game.nomorSoal][1]){
119.                        game.kedip++;
120.                        if (game.kedip > 20) game.kedip = 1;
121.                        if (game.kedip < 10){
122.                              return dataGambar.bgJawab2;
123.                        }else{
```

```
                                return dataGambar.bgJawab1;
                        }
                }else{
                        return dataGambar.bgJawab3;
                }
        }
}

function cekJawaban(num){
        if (game.stat == 1){
                game.kedip = 0;
                game.timer = 0;
                //benar
                if(game.tempJawaban[num-1] ==
                   game.tempSoal[game.nomorSoal][1]){
                        game.stat = 2;
                        game.nilai+=10;
                        //suara
                        mainkanSuara(dataSuara.benar);
                }else{
                        game.stat = 3;
                        mainkanSuara(dataSuara.salah);
                }
        }
}

function acakSoal(){
        game.nomorSoal = 0;
        game.tempSoal = [];
        game.tempJawaban = [];
        game.acak = 0;
        game.nilai = 0;
        game.stat = 1;
        game.soalMaks = 10;
        //mengacak soal, sebelumnya Array perlu di duplikat
        game.tempSoal = cloneArray(soal);
        game.tempSoal = acakArray(game.tempSoal);
}

function acakJawaban(){
        //acak jawaban
        if (game.acak == 0){
                game.acak = 1;
                game.tempJawaban =
                game.tempSoal[game.nomorSoal].slice(1, 5);
```

```
167.                    game.tempJawaban = acakArray(game.tempJawaban);
168.              }
169.        }
170.
171.        function halamanNilai(){
172.              hapusLayar();
173.              gambarFull(dataGambar.bg2);
174.              tampilkanGambar(dataGambar.bgScore, game.lebar/2,
                                  game.tinggi/2);
175.              //tampilkan nilai
176.              teks("Nilai", game.lebar/2, game.tinggi/2-25,
                        "Calibri-bold-25pt-center");
177.              teks(String(game.nilai), game.lebar/2, game.tinggi/2+40,
                        "Calibri-bold-50pt-center-merah|hitam");
178.              var retryBtn = tombol(dataGambar.ulang, game.lebar/2,
                        game.tinggi/2+125);
179.              if (tekan(retryBtn)){
180.                    acakSoal();
181.                    jalankan(gameCover);
182.              }
183.              resizeBtn(60,60);
184.        }
```

Simpan *file* **game.js** dan jalankan *file* **index.html** pada *web browser*. Apabila tidak ada kesalahan, maka *game* kuis akan dapat berjalan dengan fitur yang menyerupai *game* kuis berbasis Adobe Animate/Flash.

Gambar 102. Hasil *game* kuis

Board Game

Sebuah permainan papan (board game) pada umumnya memiliki elemen keberuntungan, strategi dan aturan khusus yang menjadikannya menarik untuk dimainkan secara terus menerus

BAB 11

GAME PAPAN (*BOARD GAME*)

11.1 Konsep *Game* Papan

Permainan papan atau diistilahkan sebagai *board game* biasanya menggunakan pion tertentu yang dipindahkan atau ditempatkan pada papan yang telah didesain dengan tampilan permainan. Sering kali permainan papan menyertakan elemen tambahan seperti kartu, properti, dadu dan sebagainya.

Permainan papan pada umumnya menampilkan kompetisi antara dua atau lebih pemain. Permainan seperti catur berakhir ketika salah satu pemain menangkap semua bidak lawan, permainan ular tangga, ludo atau *game* of life berakhir jika menyentuh petak terakhir. Sementara permainan seperti monopoli, dan sejenisnya memiliki satu set peraturan kompleks yang akan berakhir jika salah satu pemain mendominasi permainan dari segi perolehan nilai.

Gambar 103. Permainan papan ular tangga

Membuat permainan papan pada dasarnya memiliki beberapa prinsip dasar yang dapat diimplementasikan secara umum, yaitu :

1. Tujuan permainan

   Seperti pada penjelasan di atas, tujuan dari permainan dapat bervariasi dari yang paling sederhana yaitu mencapai titik tertentu, menguasai seluruh properti lawan, mendapatkan nilai tertinggi atau menyelesaikan misi khusus.

2. Menentukan Giliran dan langkah.

   Pada permainan papan yang dimainkan lebih dari satu pemain, giliran dapat ditentukan berdasarkan urutan pemain (sesuai lokasi tempat duduk atau sesuai pengacakan awal). Sedangkan untuk menentukan langkah dapat digunakan dadu, roda berputar atau jumlah langkah maksimal.

3. Desain papan

   Desain papan dibuat secara spesifik untuk memfasilitasi permainan. Dalam *game* papan seperti catur atau go, desain papan relatif sederhana yaitu kotak-kotak dalam format baris atau kolom. Sementara dalam *game* seperti ular tangga, monopoli, *game of life* dibuat secara berurutan dari angka terkecil menuju angka terbesar, dan terkadang memiliki fitur pengulangan (kembali lagi ke posisi awal).

4. Aturan khusus

   Aturan khusus dalam permainan papan sangat beragam dan disepakati antar pemain sejak awal. Pada *game* klasik catur atau go, peraturan telah ditetapkan sejak awal ditemukannya permainan dan tidak mengalami perubahan karena dinilai sudah "sempurna". Sementara pada *game* papan baru, peraturan dapat dieksplorasi sedemikian rupa untuk mendapatkan pengalaman bermain yang menarik. Pada *game of life* misalnya, pemain akan dihadapkan pada beberapa pilihan di setiap "persimpangan jalan kehidupan", meskipun terdapat elemen keberuntungan elemen strategi juga akan menjadi poin penting yang menentukan bagaimana seharusnya aturan khusus dibuat.

11.2 Aset Visual *Game* Papan

Pada bab ini akan dibuat **tutorial- 5**, yaitu sebuah *game* papan sederhana. Tujuan dari tutorial ini adalah untuk memahami prinsip kerja menggerakkan pemain di atas arena permainan. *Game* akan diberi judul "*Astro expedition*" dengan *gameplay* sederhana dimana setiap langkah pemain akan melempar dadu. Angka yang dimunculkan dadu akan dijadikan acuan dalam gerakan pemain, dan permainan akan berhenti ketika pemain mencapai petak terakhir.

Untuk membuat desain papan, dapat digunakan Aplikasi grafis seperti Adobe Photoshop, Illustrator atau Corel Draw. Papan dibuat dengan ukuran 1200x600 *pixel* dengan 32 petak. Pada tahapan ini anda dapat menyesuaikan dengan desain yang lain sesuai kebutuhan. Adapun desain papan dibuat dengan tema luar angkasa sebagai berikut :

Gambar 104. Desain papan permainan (**bgBoard**Game.**jpg**)

Aset visual selanjutnya yang diperlukan dalam pembuatan *game* papan adalah bidak pemain, yang dalam hal ini berupa *spritesheet* karakter astronot yang sedang berjalan dengan 4 arah. *Spritesheet* karakter seperti gambar di bawah ini, merupakan struktur standar *spritesheet* yang digunakan dalam game. *Spritesheet* berukuran 96x128 *pixel* dengan masing-masing sprite berukuran 32x32 *pixel*. Anda juga dapat menyesuaikan gambar bidak dengan gambar lain.

Gambar 105. *Spritesheet* **astro.png**

Untuk melakukan jumlah gerakan bidak pemain dilakukan pengacakan melalui lemparan dadu. Dadu dibuat dengan sudut pandang isometris, dan ketika diacak akan ditampilkan secara berulang-ulang dengan gambar yang berbeda, sehingga akan terbentuk efek perguliran dadu. Tampilan *spritesheet* gambar dadu adalah sebagai berikut :

Gambar 106. *spritesheet* **dadu.png**

Aset visual lainnya yang diperlukan dalam pembuatan *game* papan ini adalah sebuah tombol untuk mengacak dadu, serta sebuah cover permainan untuk menggantikan *file* cover dari *folder template*.

Gambar 107 . Cover permainan papan *Astro expedition*

11.3 Menentukan Koordinat Petak Permainan

Apabila kita perhatikan, desain area permainan pada gambar di atas sangat dinamis dimana masing-masing petak memiliki koordinat yang "berserakan", berbeda dengan papan permainan seperti catur atau ular tangga yang dapat didefinisikan dengan sederhana berdasarkan kolom dan baris petak. Sementara itu untuk mengatur pergerakan bidak pemain nantinya koordinat masing-masing petak harus dapat diidentifikasi untuk mempermudah perhitungan gerakan. Oleh karena itu koordinat masing-masing petak harus dipetakan.

Memetakan koordinat petak dapat dilakukan secara manual, dengan meletakkan gambar di layar *canvas*, kemudian menghitung posisi x dan y masing-masing petak. Namun cara ini akan memakan banyak waktu, sehingga akan lebih mudah jika kita membuat aplikasi sederhana untuk menentukan titik kordinat masing-masing petak. Perhatikan langkah berikut :

1. Duplikasi *folder* **template** dan ubah namanya menjadi *folder* **tutorial-5**.
2. Letakkan seluruh aset visual ke dalam *folder* **assets**, sehingga dalam *folder* **assets** akan terdapat *file-file* sebagai berikut :

Gambar 108. *File* di dalam *folder* assets

3. Buka *file* **game.js**, kemudian lakukan perubahan menjadi sebagai berikut :

```
1.  setGame("1200x600");
2.  game.folder  = "assets";
3.  //file gambar yang dipakai dalam game
4.  var gambar = {
5.          logo:"logo.png",
6.          startBtn:"tombolStart.png",
7.          cover:"cover.jpg",
8.          playBtn:"btn-play.png",
9.          maxBtn:"maxBtn.png",
10.         minBtn:"minBtn.png",
11.         bg:"bgBoardGame.jpg",
12.         astro:"astro.png"
13. }
14. //file suara yang dipakai dalam game
15. var suara = {
16. }
17.
18. //load gambar dan suara lalu jalankan startScreen
19. loading(gambar, suara, startScreen);
20.
21. function startScreen(){
22.         hapusLayar("#67d2d6");
23.         tampilkanGambar(dataGambar.logo, 600, 250);
24.         var startBtn = tombol(dataGambar.startBtn, 600, 350);
25.         if (tekan(startBtn)){
26.                 jalankan(halamanCover);
27.         }
28. }
29. function halamanCover(){
30.         hapusLayar("#67d2d6");
31.         gambarFull(dataGambar.cover);
32.         var playBtn = tombol(dataGambar.playBtn, 1100, 500);
33.         if (tekan(playBtn)){
34.                 setAwal();
35.                 jalankan(gameLoop);
36.         }
37.         resizeBtn(1150,50);
38. }
39.
40. function setAwal(){
41.         game.spot = "[";
42.         game.titik = 0;
43.         gambarFull(dataGambar.bg);
```

```
44. }
45.
46. function gameLoop(){
47.         if (game.mouseDitekan){
48.                 game.mouseDitekan = false;
49.                 kotak(game.mouse.x, game.mouse.y, 25, 25, 1, "red", "red");
50.                 game.spot+="["+(game.mouse.x+10)+"
    ,"+(game.mouse.y+10)+"],";
51.                 game.titik++
52.                 if (game.titik>31) {
53.                         game.spot+="]";
54.                         trace(game.spot);
55.                 }
56.         }
57. }
```

4. Simpan *file* **game.js**, kemudian jalankan *file* **index.html** melalui *intenet browser*.

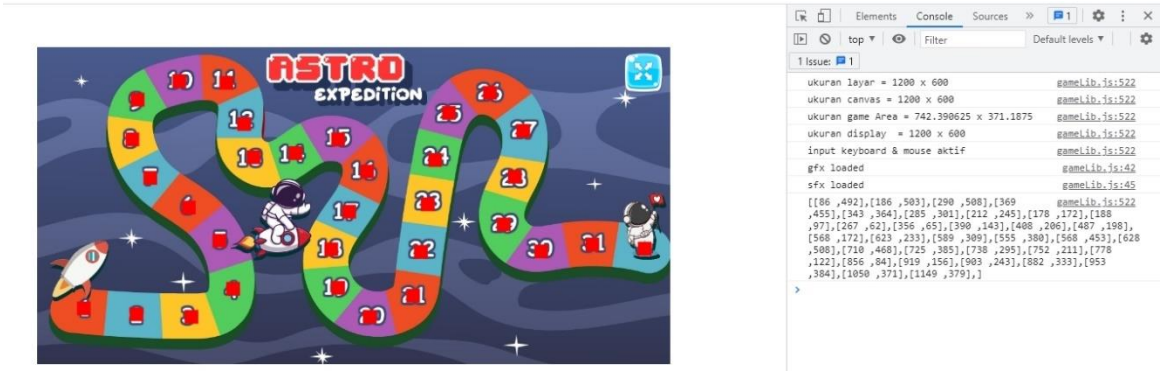
Pada tahapan ini kita perlu menentukan koordinat masing-masing petak dengan cara meng"klik" petak. Ketika klik *mouse* (lihat baris 47), maka akan diletakkan sebuah indikator kotak berwarna merah, dan kode akan mencatat koordinat tersebut dalam format *String* (baris 50). Pastikan anda membuka panel *console* (*inspect* > *console*), agar dapat mengetahui hasil pencatatan koordinat nantinya.

Gambar 109. Menentukan koordinat masing-masing petak

Klik masing-masing petak sampai dengan selesai (petak ke 32). Apabila anda memiliki desain yang berbeda, anda dapat mengubah jumlah petak pada baris ke 52 :

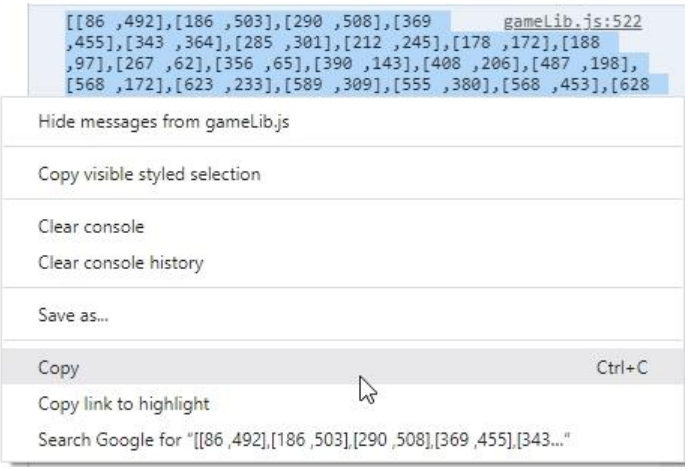
```
if (game.titik>31)
```

Ketika seluruh titik koordinat ditentukan, maka pada panel *console* akan muncul variabel koordinat petak dalam format *Array*. Pada titik ini apabila anda melakukan kesalahan penempatan, anda harus mengulang (*refresh*) *web browser* dan memulai lagi.



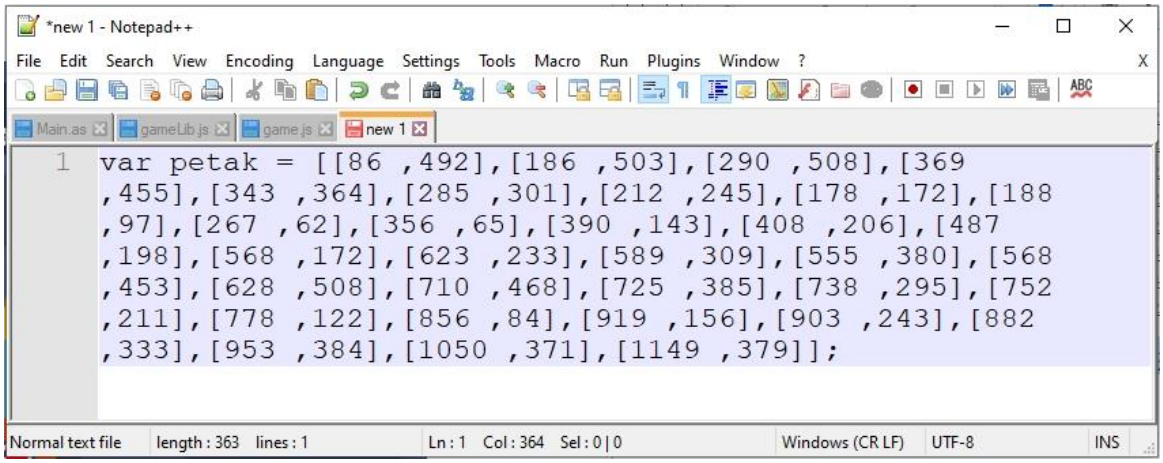
Gambar 110. Hasil trace koordinat petak

Klik kanan pada hasil *trace* tersebut kemudian pilih opsi *Copy (Ctrl+C)* untuk meng*copy* nya. Buka aplikasi Notepad++ dan *paste*kan ke *file* baru. Teknik ini sangat sederhana, dan mungkin dianggap terlalu dasar, namun demikian berdasarkan pengalaman selama beberapa tahun, teknik *tracing* koordinat seperti ini relatif mudah untuk dipelajari bagi pemula.



Gambar 111. Proses *copy* dari panel *console*

Apabila anda perhatikan hasil *copy paste* masih belum dapat dianggap sebagai variabel *Array*. Oleh karena itu perlu dilakukan sedikit pengeditan. Pertama perlu ditambahkan deklarasi `var petak =` diawal *array* agar nantinya bisa diidentifikasi. Selanjutnya pada baris terakhir perlu dilakukan penghapusan karakter , tepat sebelum karakter] penutup. Sehingga kode pada Notepad menjadi seperti pada gambar berikut :



Gambar 112. Pengeditan variabel petak

Pada variabel tersebut terdapat 32 data (sesuai jumlah petak) dalam format *Array* dimana data [0] merupakan koordinat x dan data[1] merupakan koordinat y.

12.4 Pemrograman Gerakan Bidak pada Papan

Sesuai penjelasan sebelumnya bahwa tujuan dari tutorial ini adalah untuk memahami prinsip kerja menggerakkan pemain di atas arena permainan. Maka menggerakkan bidak pada papan merupakan inti utama yang harus diprogram. Untuk melakukannya buka kembali *file* **game.js**, kemudian ubah fungsi `setAwal` dan fungsi `gameLoop`, serta tambahkan beberapa kode menjadi sebagai berikut :

```
1.  setGame("1200x600");
2.  game.folder  = "assets";
3.  //file gambar yang dipakai dalam game
4.  var gambar = {
5.          logo:"logo.png",
6.          startBtn:"tombolStart.png",
7.          cover:"cover.jpg",
8.          playBtn:"btn-play.png",
9.          maxBtn:"maxBtn.png",
10.         minBtn:"minBtn.png",
11.         bg:"bgBoardGame.jpg",
12.         astro:"astro.png",
13.         tombolAcak:"tombolAcak.png",
14.         dadu:"dadu.png"
15. }
16. //file suara yang dipakai dalam game
17. var suara = {
18. }
19.
```

```
var petak = [[86 ,492],[186 ,503],[290 ,508],[369 ,455],[343 ,364],[285
,301],[212 ,245],[178 ,172],[188 ,97],[267 ,62],[356 ,65],[390 ,143],[408
,206],[487 ,198],[568 ,172],[623 ,233],[589 ,309],[555 ,380],[568
,453],[628 ,508],[710 ,468],[725 ,385],[738 ,295],[752 ,211],[778
,122],[856 ,84],[919 ,156],[903 ,243],[882 ,333],[953 ,384],[1050
,371],[1149 ,379]];

//load gambar dan suara lalu jalankan startScreen
loading(gambar, suara, startScreen);

function startScreen(){
        hapusLayar("#67d2d6");
        tampilkanGambar(dataGambar.logo, 600, 250);
        var startBtn = tombol(dataGambar.startBtn, 600, 350);
        if (tekan(startBtn)){
                jalankan(halamanCover);
        }
}
function halamanCover(){
        hapusLayar("#67d2d6");
        gambarFull(dataGambar.cover);
        var playBtn = tombol(dataGambar.playBtn, 1100, 500);
        if (tekan(playBtn)){
                setAwal();
                jalankan(gameLoop);
        }
        resizeBtn(1150,50);
}

function setAwal(){
        game.aktif = true;
        game.menang = false;
        //posisi
        game.posisi = 0;
        game.posisiSelanjutnya = 0;
        //karakter astro
        game.hero = setSprite(dataGambar.astro, 32, 32);
        game.hero.x = petak[game.posisi][0];
        game.hero.y = petak[game.posisi][1];
        game.skalaSprite = 2;
        game.hero.frame = 8;
        //dadu
        game.dadu = setSprite(dataGambar.dadu, 64, 64);
}

```

```
60. function gameLoop(){
61.         hapusLayar();
62.         gambarFull(dataGambar.bg);
63.         //menampilkan astro
64.         sprite(game.hero);
65.         //tombol acak
66.         if (game.aktif && !game.menang){
67.                 var acak = tombol(dataGambar.tombolAcak, 1050, 550);
68.                 if (tekan(acak)){
69.                         acakDadu();
70.                 }
71.         }else{
72.                 gerakDadu();
73.                 gerakHero();
74.                 sprite(game.dadu);
75.         }
76. }
77.
78. function acakDadu(){
79.         game.aktif = false;
80.         game.dadu.x = 100+acak(1000);
81.         game.dadu.y = 100+acak(400);
82.         game.dadu.frame = acak(6)+1;
83.         game.dadu.delay = 1;
84. }
85.
86. function gerakDadu(){
87.         if (game.posisi != game.posisiSelanjutnya || game.menang)return;
88.         game.dadu.timer++;
89.         if (game.dadu.timer>game.dadu.delay){
90.                 game.dadu.timer = 0;
91.                 game.dadu.delay+=0.5;
92.                 //acak dadu
93.                 game.dadu.frame = acak(6)+1;
94.                 if (game.dadu.delay>5){
95.                         game.posisiSelanjutnya = game.posisi+game.dadu.frame;
96.                         //sampai di akhir
97.                         if (game.posisiSelanjutnya>=petak.length-1){
98.                                 game.posisiSelanjutnya = petak.length-1
99.                         }
100.                        }
101.                }
102.        }
103.
```

```
104.      function gerakHero(){
105.            if (game.posisi != game.posisiSelanjutnya){
106.                  //hitung sudut ke petak selanjutnya
107.                  var arah = sudut(petak[game.posisi][0],
                                       petak[game.posisi][1],
                                       petak[game.posisi+1][0],
                                       petak[game.posisi+1][1]);
108.                  //gerakkan hero
109.                  game.hero.x += Math.cos(arah*3.14159265359/180);
110.                  game.hero.y += Math.sin(arah*3.14159265359/180);
111.                  aturFrameHero(arah);
112.                  //sudah mendekati petak
113.                  if (jarak(petak[game.posisi+1][0],
                                petak[game.posisi+1][1],
                                game.hero.x, game.hero.y) < 10){
114.                        game.posisi++;
115.                        if (game.posisi == game.posisiSelanjutnya){
116.                              //sudah sampai di lokasi
117.                              //cek apakah sampai di finish
118.                              if (game.posisi >= petak.length-1){
119.                                    trace("menang");
120.                                    game.menang = true;
121.                                    return;
122.                              }
123.                              game.aktif = true;
124.                        }
125.                  }
126.            }
127.      }
128.
129.      function aturFrameHero(arah){
130.            var dir=1;
131.            //menetralisis sudut agar rentang 0-360
132.            //menghitung arah dimulai dari sudut 0 = jam 3
133.            if (arah < 0) arah+=360;
134.            if (arah > 135 && arah < 225) dir = 2;
135.            if (arah > 225 && arah < 315) dir = 4;
136.            if (arah > 45 && arah < 135) dir = 1;
137.            if (arah > 315 || arah < 45) dir = 3;
138.            game.hero.timer++;
139.            if (game.hero.timer > game.hero.delay){
140.                  game.hero.timer = 0;
141.                  game.hero.step++;
142.                  if (game.hero.step > 3) game.hero.step = 1;
143.                  game.hero.frame= (dir-1)*3+game.hero.step;
```

```
144.             }
145.         }
```

Simpan *file* **game.js**, kemudian jalankan *file* **index.html** melalui *intenet browser*. Pada titik ini permainan akan bisa dimulai dengan menekan tombol “acak dadu”, dadu akan muncul secara acak dan ketika berhenti karakter Astronot akan berjalan menuju ke petak selanjutnya sesuai dengan angka yang muncul di dadu. Permainan akan berakhir ketika pemain menyentuh petak terakhir, dan pada panel console akan muncul teks “menang”.

Gambar 113. Hasil permainan papan

Penjelasan Program

Variabel `petak` yang merupakan hasil dari *copy paste* koordinat petak diletakkan pada baris ke 20 setelah deklarasi variabel `gambar` dan variabel `suara`. Pada fungsi `setAwal` dilakukan pengaturan dasar seperti variabel `posisi` saat ini dan `posisiSelanjutnya`, karakter pemain (`game.hero`), dan pengaturan dadu (`game.dadu`).

Pada awal fungsi `gameLoop` gambar papan permainan ditampilkan secara penuh ke *canvas* menggunakan kode :

```
62.         gambarFull(dataGambar.bg);
```

yang selanjutnya diikuti secara berturut-turut dengan menampilkan *sprite* `hero`, tombol acak dan *sprite* `dadu`. Pada baris ke 66 terdapat kondisi `if` yang mempersyaratkan *game* aktif dan belum dimenangkan agar tombol acak muncul di layar

```
66.         if (game.aktif && !game.menang){
```

ketika tombol acak ditekan, maka fungsi `acakDadu` akan dijalankan. Fungsi `acakDadu` sendiri relatif sederhana, pada awalnya diatur posisi keluarnya dadu secara acak, dan *frame* dadu diacak untuk mengeluarkan angka 1-6.

Kembali ke fungsi `gameLoop`, ketika `acakDadu` dieksekusi maka `game.aktif` bernilai *false*. Hal ini akan menyebabkan fungsi `gerakDadu` dan `gerakHero` dijalankan (baris 72 dan 73). Fungsi `gerakDadu` akan dijalankan terlebih dahulu karena kondisi `game.posisi != game.posisiSelanjutnya`. Saat ini nilai kedua variabel tersebut sama, sehingga kode baris 88-100 akan dijalankan. Dadu akan berubah-ubah *frame* nya sesuai dengan variabel `timer`. Perubahan *frame* melalui fungsi `acak`, akan menghasilkan efek dadu bergulir. Selanjutnya pada titik tertentu dadu akan berhenti, dan nilai dari *frame* dadu tersebut ditambahkan dengan variabel posisi untuk menentukan `posisiSelanjutnya`.

```
95. game.posisiSelanjutnya = game.posisi+game.dadu.frame;
```

Dengan demikian fungsi `gerakDadu` akan berhenti dan fungsi `gerakHero` akan aktif. Gerakan `hero` sedikit kompleks untuk dijelaskan. Pada awalnya digunakan fungsi `sudut` untuk mencari sudut antara petak posisi saat ini dan petak 1 langkah ke depan.

```
sudut(kordinat X awal, kordinat Y awal,
      kordinat X selanjutnya, kordinat Y selanjutnya);
```

Fungsi ini akan menghasilkan sudut dalam format -180 sampai 180 derajat (standar perhitungan sudut pada program komputer). Sudut ini dapat digunakan untuk mengatur gerakan pemain dengan menggunakan rumus trigonometri sebagai berikut :

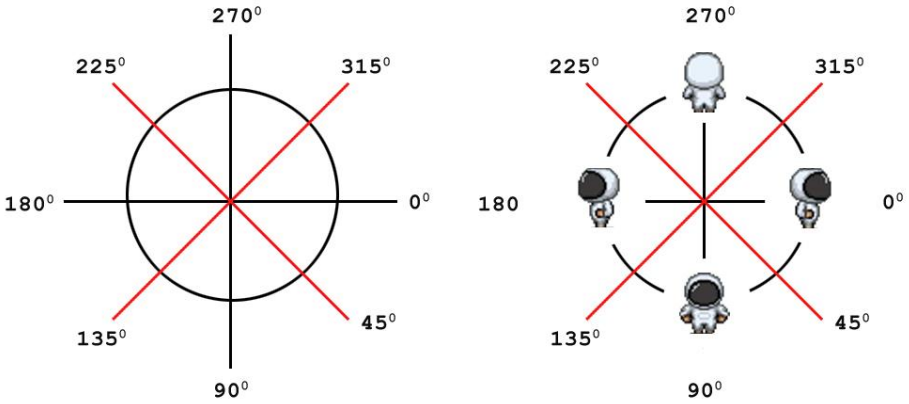
```
109.      game.hero.x += Math.cos(arah*Math.PI/180);
110.      game.hero.y += Math.sin(arah*Math.PI/180);
```

Pergerakan sumbu x adalah sejauh *cosinus* dari sudut dikalikan satu derajat *radian* (pi dibagi 180), dan y adalah *sinus* dari perkalian sudut dan satu derajat *radian*. Dengan demikian karakter pemain akan bergeser seiring waktu mendekati titik koordinat petak berikutnya.

Untuk mengatur *frame* mana yang akan muncul (menganimasikan karakter astronot), digunakan kode :

```
111.      aturFrameHero(arah);
```

Fungsi `aturFrameHero` digunakan untuk mengarahkan ke *frame* yang tepat sesuai dengan sudut arah gerakan. Perhatikan gambar berikut :



Gambar 114. Pengaturan *frame sprite* berdasarkan sudut

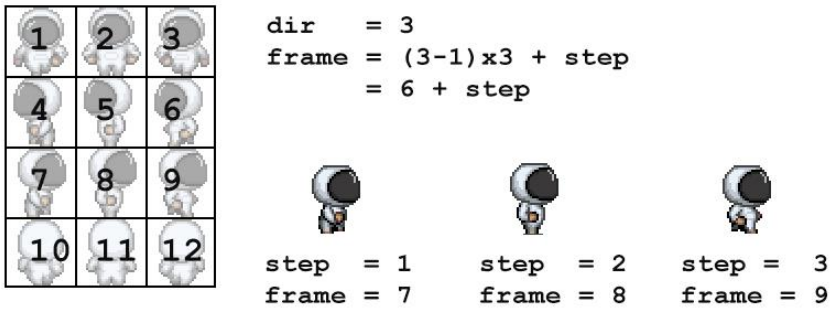
Perhitungan sudut 0 derajat berada pada arah sudut jam 3, dan bergerak searah jarum jam. Sudut bernilai -180 sampai 180 sehingga harus dijadikan format 0-360 dengan menambahkan 360 pada sudut yang bernilai negatif.

```
133.      if (arah < 0) arah+=360;
```

Selanjutnya terkait dengan *sprite sheet* astronot yang digunakan, yaitu memiliki 4 arah gerak, maka kuadran dibagi menjadi 4 bagian masing-masing 90 derajat dalam posisi diagonal (lihat gambar kanan). Dengan pembagian kuadran tersebut ditentukan variabel `dir` dan variabel `step` yang selanjutnya digunakan untuk menentukan *frame* mana yang ditampilkan untuk menghasilkan animasi yang sesuai dengan arah gerak karakter.

```
143.      game.hero.frame= (dir-1)*3+game.hero.step;
```

Sebagai contoh ketika variabel `dir` bernilai 3 (karakter ke kanan), maka *frame* akan bernilai `(3-1) x 3 + step` atau bernilai `6` + variabel `step`. Variabel `step` bernilai antara 1 dan 3, hal ini ditentukan dari jumlah *frame* yang membentuk animasi gerak berjalan. Dalam *sprite sheet* astro tersebut, masing masing arah memilik 3 gerakan. Ketika step bernilai 1, maka frame akan bernilai 7. Ketika step bernilai 2 frame akan bernilai 8, dan seterusnya sampai kembali lagi ke step 1 akibat kode `if (game.hero.step > 3) game.hero.step = 1;` dengan teknik ini karakter akan memiliki animasi berjalan ke arah kanan.



Gambar 115. Contoh perhitungan frame karakter

Ketika karakter `hero` bergerak, jarak antara posisi `hero` dengan koordinat titik petak selanjutnya dihitung dengan menggunakan fungsi `jarak`.

```
113.      if (jarak(petak[game.posisi+1][0], petak[game.posisi+1][1],
              game.hero.x, game.hero.y) < 10){
```

Ketika karakter `hero` sudah berada pada titik koordinat, hal ini berarti karakter `hero` telah melangkah 1 petak dan siap melanjutkan ke petak selanjutnya sampai dengan nilai variabel `petakSelanjutnya` yang ditentukan oleh lemparan dadu. Proses ini akan berulang dan ketika `hero` sudah sampai pada petak yang sesuai dengan variabel `petakSelanjutnya` maka variabel `game.aktif` kembali bernilai *true* dan tombol acak akan muncul kembali.

```
115.      if (game.posisi == game.posisiSelanjutnya){ … }
```

Ketika tombol acak ditekan kembali, maka proses akan berulang sampai dengan pemain menyentuh petak terakhir dan variabel `game.menang` akan bernilai *true*.

12.5 Tahapan Selanjutnya

Pada titik ini tujuan dari **tutorial-5** telah tercapai. Konsep dasar menggerakkan bidak pemain di atas permainan papan telah dijelaskan. Tahapan selanjutnya yang harus dikembangkan sendiri oleh pembaca adalah menambahkan beberapa hal agar permainan papan ini menjadi lebih menarik, lebih interaktif dengan aturan-aturan yang ditetapkan. Contoh teknik penambah fitur yang dapat dilakukan adalah :

1. Menambahkan pemain ke dua dan seterusnya

   Untuk menambahkan pemain lain terdapat 2 point penting yang harus ditambahkan. Pertama, *spritesheet* yang berbeda untuk membedakan antara pemain satu dan pemain lainnya, hal ini tentusaja akan berimbas pada fungsi `aturFrameHero`. Kedua adalah penambahan variabel `giliran` dan variabel `jumlahPemain`. Pada baris 123 setelah

pemain sampai di petak tujuan, maka variabel `giliran` akan ditambah 1, dan jika variabel `giliran` lebih besar dari variabel `jumlahPemain`, maka variabel `giliran` dikembalikan lagi ke pemain pertama.

```
124.      game.giliran++;
125.      if (game.giliran > game.jumlahPemain) game.giliran = 1;
```

2. Menambahkan kuis atau tantangan

   Untuk menambahkan kuis dapat diintegrasikan dengan **tutorial-4**, yang mana ketika pemain sampai pada petak tujuan maka kuis akan dimunculkan dan giliran pemain baru akan dimulai lagi setelah kuis terjawab. Hal ini mengharuskan anda mengedit pada kode baris 115 setelah pemain selesai melangkah, dan mengharuskan penambahan fungsi `kuis` serta mengatur giliran selanjutnya setelah kuis terjawab.

   Pada fungsi `gameLoop`

```
115.            if (game.posisi == game.posisiSelanjutnya){
116.                  //sudah sampai di lokasi lalu tampilkan kuis
117.                  tampilkanKuis();
```

   Pada fungsi `tampilkanHasil` pada tutorial kuis

```
146.            function tampilkanHasil(){
147.                  . . .
148.                  //restart soal
149.                  if (game.timer>120){
150.                        game.stat = 1;
151.                        game.giliran++;
152.                        game.aktif = true;
```

3. Penambahan bonus naik atau turun

   Seperti halnya pada *game* ular tangga, anda dapat menambahkan bonus naik ke petak yang lebih jauh atau turun ke petak sebelumnya. Untuk melakukannya dapat digunakan sebuah variabel bertipe *Array* yang menunjukkan nilai petak awal dan petak akhir, misal

```
var  petakBonus = [[5, 12], [13, 24], [27, 15]];
```

   Ketika pemain sampai pada petak, giliran tidak langsung dilanjutkan namun dilakukan pengecekan `petakBonus` terlebih dahulu. Sebagai contoh ketika bidak pemain berhenti di petak ke 5, sesuai variabel `petakBonus` tersebut, maka pemain akan naik ke petak 12

dan barulah giliran dilanjutkan ke pemain selanjutnya. Tutorial ini dapat anda *porting* dari tutorial *game* ular tangga yang ada di *channel* Youtube wandahw.

4. Menambahkan halaman pemenang

   Pada **tutorial-5** di atas, indikator kemenangan hanya berupa `trace` "menang" pada panel *console*. Anda dapat mengarahkan kemenangan ke halaman khusus (fungsi khusus) dengan cara mengganti kode trace menang dengan kode :

```
jalankan(halamanMenang);
fungsi halamanMenang() {
   //tampilkan antar muka untuk menunjukkan pemenang'
   . . .
}
```

5. Penambahan fitur/aturan lainnya

   Anda dapat menambahkan elemen tambahan lainnya seperti suara, efek partikel, dan sebagainya.

Komersialisasi

Pasar game berkembang pesat setiap tahunnya Peluang mengkomersialisasikan game HTML 5 cukup besar. Game HTML 5 dapat dijadikan web game maupun mobile game, dipublish sendiri maupun dilisensikan ke publisher lain

BAB 12
PENUTUP

12.1 Pengembangan Lanjutan

Setelah memahami beberapa konsep dasar pengembangan *game* dengan menggunakan *Javascript* tahapan selanjutnya adalah mengembangkan aplikasi yang lebih kompleks. Pada saat ini sumber belajar sangat melimpah dan dapat diakses dengan mudah, oleh karena itu tidak ada alasan untuk tidak memperdalam keilmuan kita terkait pengembangan *game* dan aplikasi.

Game HTML5 *canvas* dengan bahasa pemrograman *javascript* memiliki keunggulan yaitu dapat dijalankan melalui *web browser* tanpa plugin serta dapat dikembangkan dengan aplikasi gratis. Dengan cepat HTML5 menggeser *Flash Player* sebagai standar interaktivitas aplikasi berbasis web, dengan kata lain mempelajari HTML5 *canvas* dan *javascript* berarti mempelajari suatu teknologi yang sifatnya kekinian (*uptodate*).

Game HTML 5 juga dapat di*publish* ke format Android ataupun iOS, hal ini berarti pasar game HTML 5 sangat terbuka lebar, baik dirilis sendiri dengan pendapatan berasal dari iklan atau penjualan dalam aplikasi (*in Apps Purchase*), atau dirilis melalui publisher lain melalui format lisensi atau beli putus. Apapun metodenya, mengkomersialisasikan sebuah game HTML5 adalah sebuah peluang yang cukup baik untuk dicoba.

Melalui buku ini diharapkan pembaca memiliki dasar dalam mengembangkan aplikasi HTML5 *canvas*, yang selanjutnya dapat dikembangkan lebih lanjut lagi untuk keperluan pembuatan aplikasi, *game*, maupun multimedia interaktif. Karya yang dihasilkan selanjutnya dapat digunakan untuk keperluan pribadi maupun keperluan komersial. Berdasarkan pengalaman penulis, mempelajari, membuat dan mengkomersialisasikan *game* dapat menghasilkan keuntungan yang cukup signifikan mengingat pasar *game* yang berkembang secara pesat, dan kebutuhan akan *game-game* baru yang terus bertambah. Oleh karena itu penulis berharap tutorial di dalam buku ini dapat menjadi awalan bagi para pengembang *game* di Indonesia.

Pada akhirnya penulis ingin berterimakasih kepada seluruh pembaca dan semoga buku ini ditindak lanjuti dengan banyaknya *game* HTML5 yang berkualitas. Penulis juga mengharapkan saran dan masukan terkait pengembangan *library* **gameLib.js** agar menjadi lebih efisien dalam mengembangkan *game* HTML5.

12.2 Diskusi Terkait Pengembangan Aplikasi

Penulis memberikan kesempatan kepada para pembaca untuk berdiskusi terkait pengembangan aplikasi, *game* atau media digital lainnya melalui beberapa platform sebagai berikut :

1. Email wandah@wandah.com
2. Kolom komentar pada Youtube Channel : www.youtube.com/user/wandahw
3. Pesan Whatsapp
4. Kolom komentar pada tutorial di situs www.wandah.org

TENTANG PENULIS

Wandah Wibawanto S.Sn., M.Ds. lahir di Malang, 28 Januari 1983. Menyelesaikan program S1 Desain Komunikasi Visual di Universitas Negeri Malang pada tahun 2006 dan S2 Magister Desain (*Game tech*) ITB pada tahun 2012 dan saat buku ini ditulis sedang menempuh program S3 Pendidikan Seni Universitas Negeri Semarang. Mendalami pengembangan *game* Flash sejak tahun 2001 dan mengkhususkan pada pengembangan *game browser*.

Pada tahun 2006 bekerja di sebuah perusahaan *game* asing yaitu FreeOnline *Game*s.com dan menghasilkan beberapa karya *game* yang cukup populer, di antaranya adalah Sim Taxi, Gangster Life, The Empire, dan beberapa *game* lainnya. Tahun 2008 memutuskan untuk mengembangkan *game* flash secara independen dan menghasilkan beberapa *game* flash yang ditujukan untuk beberapa *game* portal seperti Armor*game*s, *Game*ninja, *Game*books, DailyFree*Game*s, dan Froyo*game*s.com. Mendapatkan beberapa penghargaan di bidang aplikasi *game*s seperti pemenang INAICTA 2009 kategori digital media, *Honorable mention Dictionary Games* award, FOG *daily game programing challenge* dan beberapa lainnya. Tahun 2014 bergabung menjadi dosen DKV Universitas Negeri Semarang, menjadi peneliti di bidang ekonomi kreatif, dan pengembangan media edukasi.

Beberapa buku yang telah ditulis terkait dengan pengembangan Aplikasi antara lain :

- Membuat *Game* dengan Flash (2003) Penerbit Andi, Yogya
- Dasar-dasar Pemrograman Flash *Game* (2006) e-book
- Membuat Flash *Game* 3D (2013) Penerbit Andi, Yogya
- Desain dan Pemrograman Multimedia Interaktif (2017) Penerbit Cerdas Ulet Kreatif, Jember
- Membuat bermacam-macam *Game Android* dengan Adobe Animate (2019) Penerbit Andi, Yogya
- Game Edukasi *Role Play Game* (RPG) (2020) Penerbit LPPM UNNES
- Laboratorium Virtual – Konsep dan Pengembangan Simulasi Fisika (2020) Penerbit LPPM UNNES

Beberapa buku terkait ekonomi kreatif antara lain :

- *Book chapter* "Kolase Pemikiran Ekonomi Kreatif"

- Pendampingan OPD dalam Pengembangan Ekonomi Kreatif Kab/Kota
- Integrasi Rencana Induk Pengembangan Ekonomi Kreatif (Rindekraf) dalam RPJMD Kab/Kota

Beberapa karya aplikasi, tutorial dan materi perkuliahan dapat ditemukan di situs wandah.com, wandah.org dan froyogames.com. Penulis dapat dihubungi via email di wandah@wandah.comatau wandah@mail.unnes.ac.id

DAFTAR PUSTAKA

Faas, Travis. 2017. *An Introduction to HTML5 Game Development with Phaser.js*. CRC Press : Taylor & Francis Group.

Feil, John dan Scattergood, Marc. 2005. *Beginning Game Level Design*. Boston: Thomson Course Technology PTR.

Fox, Brent. 2005. *Game Interface Design*. Boston: Thomson Course Technology PTR.

Kim, Sangkyun dkk. 2018. *Advances in Game-Based Learning: Gamification in Learning and Education*. Switzerland: Springer International Publishing.

Macklin, Colleen dan Sharp, John. 2016. *Games, Design and Play: A Detailed Approach to Iterative Game Design*. Pearson Education

McAllister, Graham dan White, Gareth R. 2015. *Game User Experience Evaluation:Video Game Development and User Experience*. Switzerland: Springer International Publishing.

Pedersen, Roger E. 2003. *Game Design Foundations*. Texas: Wordware Publishing, Inc.

Ramtal, Dev dan Dobre, Adrian. 2014. *Physics for Javascript Games, Animation, and Simulations with HTML5 Canvas*. Worthing: Apress

Spuy, Rex van der. 2012. *Foundation Game Design with Actionscript 3.0*. New York: Friendsoft-Apress.

Takatalo, Jari dkk. 2015. *Game User Experience Evaluation: Understanding Presence, Involvement, and Flow in Digital Games*. Switzerland: Springer International Publishing.

Tyers, Ben. 2017. *Learn RPGs in GameMaker: Studio Build and Design Role Playing Games*. Worthing: Apress.

Wibawanto, Wandah. 2020. *Game Edukasi RPG*. Semarang: Penerbit LPPM UNNES

Website :

www.wandah.org

www.opengameart.org

https://kenney.itch.io

https://pixelfrog-assets.itch.io/

INDEX

A

B

C

D

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

Wandah Wibawanto

DASAR PENGEMBANGAN GAME HTML 5





Buku ini membahas proses pengembangan game HTML5 dengan bahasa pemrograman Javascript. Pengembangan game dapat dilakukan dengan menggunakan aplikasi teks editor sederhana seperti Notepad++. Pembahasan dalam buku ini meliputi proses pengembangan grafis, pengkodean, dan penjelasan terkait kode yang digunakan.

Penerbit LPPM Universitas Negeri Semarang
Gedung Prof. Dr. Retno Sriningsih Satmoko,
Penelitian dan Pengabdian Masyarakat,
Kampus Sekaran, Gunungpati, Semarang
50229

ISBN 978-623-366-004-4 (PDF)