

Get Started Galaxio

Disusun oleh tim asisten Strategi Algoritma 2022 / 2023

Update:

2 Februari 2023 pukul 21.06 : Menambahkan informasi mengenai enums "PlayerAction" dan "ObjectTypes" pada starter bot.

4 Februari 2023 pukul 11.57 : Menambahkan informasi mengenai enums "Effect" dan models "GameObject" pada starterbot.

Daftar Isi:

[Gambaran](#)

[Prasyarat](#)

[Starter Pack](#)

[Cara Menjalankan Game](#)

[Starter Bot](#)

[Visualizer](#)

Gambaran

Dokumen ini dibuat dengan tujuan memberikan gambaran singkat tentang permainan *Galaxio*. Silahkan baca dan pahami dokumen untuk mengetahui hal-hal dasar yang harus dilakukan untuk memulai pengerjaan tugas besar. Dokumen ini tidak memuat seluruh informasi terkait *Galaxio*.

Prasyarat

- .NET Core 3.1
 - [Windows](#)
 - [Linux](#)
 - [MacOS](#)
-

Starter Pack

Untuk memulai, silahkan unduh dan ekstrak arsip *starter pack* yang dapat diakses pada pranala berikut: [starter pack](#). Isi dari arsip *starter pack* adalah beberapa *folder* dan *script* sebagai berikut:

```
|— engine-publish
|— logger-publish
|— reference-bot-publish
|— runner-publish
|— starter-bots
|— visualiser
|— building-a-bot.md
|— README.md
└— run.sh
```

Ada beberapa komponen yang perlu diketahui untuk memahami cara kerja game ini, mereka adalah:

- Engine
Engine merupakan komponen yang berperan dalam mengimplementasikan logic dan rules game.
- Runner
Runner merupakan komponen yang berperan dalam menggelar sebuah *match* serta menghubungkan bot dengan engine.
- Logger
Logger merupakan komponen yang berperan untuk mencatat *log* permainan sehingga kita dapat mengetahui hasil permainan. Log juga akan digunakan sebagai input dari visualizer

Garis besar cara kerja program game Galaxio adalah sebagai berikut:

1. Runner –saat dijalankan– akan meng-*host* sebuah *match* pada sebuah hostname tertentu. Untuk koneksi lokal, runner akan meng-*host* pada localhost:5000.
2. Engine kemudian dijalankan untuk melakukan koneksi dengan runner. Setelah terkoneksi, Engine akan menunggu sampai bot-bot pemain terkoneksi ke runner.
3. Logger juga melakukan hal yang sama, yaitu melakukan koneksi dengan runner.
4. Pada titik ini, dibutuhkan beberapa bot untuk melakukan koneksi dengan runner agar *match* dapat dimulai. Jumlah bot dalam satu pertandingan didefinisikan pada atribut BotCount yang dimiliki file JSON "*appsettings.json*". File tersebut terdapat di dalam folder "runner-publish" dan "engine-publish".
5. Permainan akan dimulai saat jumlah bot yang terkoneksi sudah sesuai dengan konfigurasi.
6. Bot yang terkoneksi akan mendengarkan event-event dari runner. Salah satu event yang paling penting adalah ReceiveGameState karena memberikan status game.
7. Bot juga mengirim event kepada runner yang berisi aksi bot.
8. Permainan akan berlangsung sampai selesai. Setelah selesai, akan terbuat dua file json yang berisi kronologi *match*.

Cara Menjalankan Game

Berdasarkan gambaran cara kerja program game yang telah disebutkan sebelumnya, berikut merupakan cara menjalankan game secara lokal di **Windows**:

1. Lakukan konfigurasi jumlah bot yang ingin dimainkan pada *file* JSON `"appsettings.json"` dalam folder `"runner-publish"` dan `"engine-publish"`
2. Buka terminal baru pada folder `runner-publish`.
3. Jalankan runner menggunakan perintah `"dotnet GameRunner.dll"`
4. Buka terminal baru pada folder `engine-publish`
5. Jalankan engine menggunakan perintah `"dotnet Engine.dll"`
6. Buka terminal baru pada folder `logger-publish`
7. Jalankan engine menggunakan perintah `"dotnet Logger.dll"`
8. Jalankan seluruh bot yang ingin dimainkan
9. Setelah permainan selesai, riwayat permainan akan tersimpan pada 2 *file* JSON `"GameStateLog_{Timestamp}"` dalam folder `"logger-publish"`. Kedua file tersebut diantaranya *GameComplete* (hasil akhir dari permainan) dan proses dalam permainan tersebut.

Note: Untuk menjalankan Game Runner, Engine, atau Logger pada **UNIX-based** OS dapat memodifikasi atau langsung menjalankan `"run.sh"` yang tersedia pada *starter-pack*. Pada windows dapat menggunakan atau memodifikasi batch script seperti berikut:

```
@echo off
:: Game Runner
cd ./runner-publish/
start "" dotnet GameRunner.dll

:: Game Engine
cd ../engine-publish/
timeout /t 1
start "" dotnet Engine.dll

:: Game Logger
cd ../logger-publish/
timeout /t 1
start "" dotnet Logger.dll

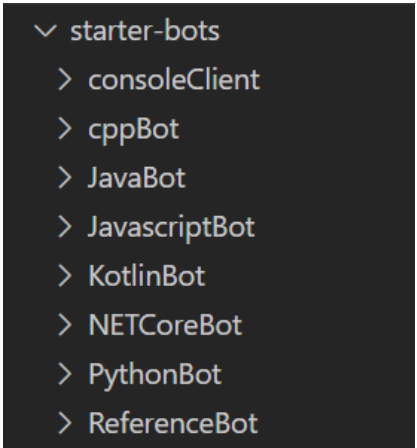
:: Bots
```

```
cd ../reference-bot-publish/  
timeout /t 3  
start "" dotnet ReferenceBot.dll  
timeout /t 3  
start "" dotnet ReferenceBot.dll  
timeout /t 3  
start "" dotnet ReferenceBot.dll  
timeout /t 3  
start "" dotnet ReferenceBot.dll  
cd ../  
  
pause
```

Bagian :: Bots dapat dimodifikasi dari menjalankan Reference Bot menjadi Bot kalian sendiri. Untuk langkah build *source code* bot kalian dan menjalankannya dapat dibaca pada bagian Starter Bot di bawah ini.

Starter Bot

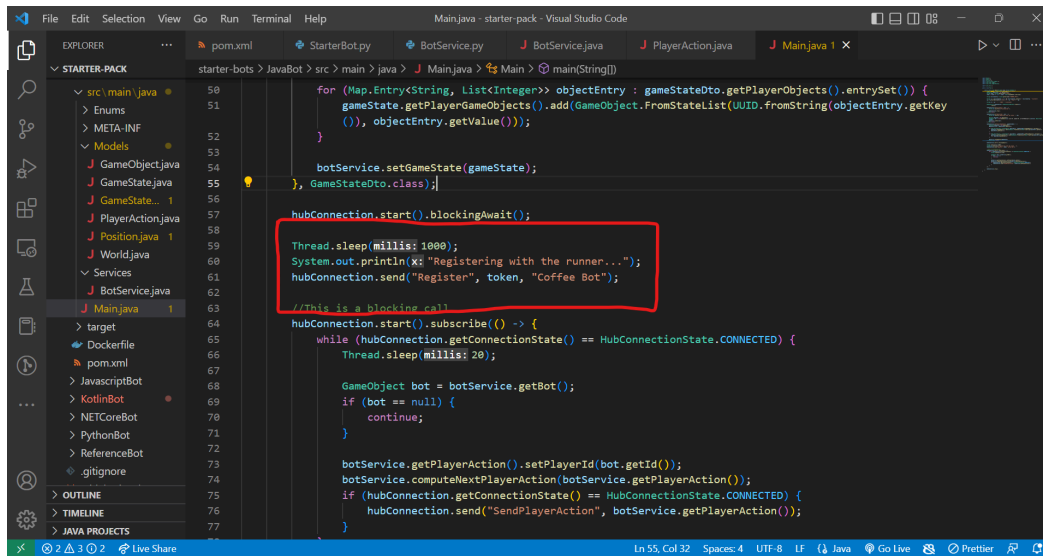
Starter pack menyediakan *starter bot* sebagai titik awal dari pengembangan bot milik kita sendiri. Starter bot sudah dibekali kode yang dibutuhkan oleh bot untuk melakukan koneksi dengan runner sehingga kita dapat fokus kepada strategi permainan. Gambar dibawah menunjukkan bahwa starter-pack memberikan beberapa pilihan bahasa untuk starter bot. Namun pada tugas besar ini, kita akan menggunakan bot yang berbahasa Java.



```
▼ starter-bots  
  > consoleClient  
  > cppBot  
  > JavaBot  
  > JavascriptBot  
  > KotlinBot  
  > NETCoreBot  
  > PythonBot  
  > ReferenceBot
```

Untuk memulai, silahkan salin folder JavaBot (atau tidak usah, terserah kalian) dan beri nama bot kalian dengan cara mengubah string “Coffee Bot” pada file Main.java baris ke-61 menjadi apapun terserah kalian. Setelah itu, buatlah sebuah repositori GitHub

agar kalian dapat bekerja sama dengan teman satu kelompok (jangan lupa private). Selamat!! Pada titik ini kalian telah memiliki bot sendiri.



```
50 for (Map.Entry<String, List<Integer>> objectEntry : gameStateDto.getPlayerObjects().entrySet()) {
51     gameState.getPlayerGameObjects().add(GameObject.FromStateList(UUID.fromString(objectEntry.getKey()), objectEntry.getValue()));
52 }
53
54 botService.setGameState(gameState);
55 }, GameStateDto.class);
56
57 hubConnection.start().blockingAwait();
58
59 Thread.sleep(millis: 1000);
60 System.out.println(x: "Registering with the runner...");
61 hubConnection.send("Register", token, "Coffee Bot");
62
63 //This is a blocking call
64 hubConnection.start().subscribe(() -> {
65     while (hubConnection.getConnectionState() == HubConnectionState.CONNECTED) {
66         Thread.sleep(millis: 20);
67
68         GameObject bot = botService.getBot();
69         if (bot == null) {
70             continue;
71         }
72
73         botService.getPlayerAction().setPlayerId(bot.getId());
74         botService.computeNextPlayerAction(botService.getPlayerAction());
75         if (hubConnection.getConnectionState() == HubConnectionState.CONNECTED) {
76             hubConnection.send("SendPlayerAction", botService.getPlayerAction());
77         }
78     }
79 }
```

Baris 73 sampai baris 77 kode Main.java menunjukkan bahwa bot mengirim aksi ke runner menggunakan metode “send” dengan argumen aksi bot. Aksi ini dihitung dan diperbarui oleh bot menggunakan metode computeNextPlayerAction milik kelas BotService. Kami menyarankan untuk mengimplementasikan algoritma greedy di dalam kelas BotService.

Peringatan:

Apabila anda melewatkan step ini, maka akan mengirimkan pesan error “MISSING VALUE” dan bot tidak bisa bergerak sesuai yang diharapkan

Kami menemukan bahwa starter bot mungkin “outdated” terhadap rules/engine rilis terakhir game. Contohnya adalah pada enum “PlayerActions” yang dimiliki starter bot hanya terdapat 4 pilihan aksi. Padahal berdasarkan [game rules](#), bot dapat melakukan 10 jenis aksi. Sama halnya dengan enum “ObjectTypes”.

Oleh sebab itu, silahkan baca dan pahami betul-betul game-rules yang ada karena mungkin ada perbedaan lain pada enums ataupun models. Sesuai dengan kode pada game engine, berikut adalah value yang sesuai:

```

1  namespace Domain.Enums
2  {
3      public enum GameObjectType
4      {
5          Player = 1,
6          Food = 2,
7          Wormhole = 3,
8          GasCloud = 4,
9          AsteroidField = 5,
10         TorpedoSalvo = 6,
11         Superfood = 7,
12         SupernovaPickup = 8,
13         SupernovaBomb = 9,
14         Teleporter = 10,
15         Shield = 11
16     }
17 }

```

16 lines (16 sloc) | 330 Bytes

```

1  namespace Domain.Enums
2  {
3      public enum PlayerActions
4      {
5          Forward = 1,
6          Stop = 2,
7          StartAfterburner = 3,
8          StopAfterburner = 4,
9          FireTorpedoes = 5,
10         FireSupernova = 6,
11         DetonateSupernova = 7,
12         FireTeleport = 8,
13         Teleport = 9,
14         ActivateShield = 10
15     }
16 }

```

Silakan tambahkan pasangan key-value baru pada enum di JavaBot untuk menyesuaikannya.

Note: Dikutip dari

<https://github.com/EntelectChallenge/2021-Galaxio/tree/develop/game-engine/Domain/Enums>

NOTE:

Apabila anda sudah menggunakan suatu command dan tidak berhasil, contoh menggunakan fire torpedoes tapi tidak menembak, kemungkinan besar karena anda menuliskan di enum FIRE_TORPEDOES(5);

Coba tidak menggunakan underscorenya, seharusnya itu udah fix problemnya.

(Hal itu karena enumnya dikirim sebagai string dan yang dibaca alfabet doang, bisa dicek di kode game runner)

Masukan dari salah satu teman kalian (request anon):

Kalau yang ini **ga wajib**, cuman membantu memberikan data tambahan

Perhatikan file models → GameObject.

```

public class GameObject {
    public UUID id;
    public Integer size;
    public Integer speed;
    public Integer currentHeading;
    public Position position;
    public ObjectTypes gameObjectType;
}

```

Sementara ini, fungsi FromStateList hanya mengambil data untuk 6 atribut (size, speed, currentHeading, gameObjectType, posisi X, posisi Y). Akan tetapi, apabila diperiksa, berikut adalah statelist yang dikirim oleh game engine untuk objek player.

```

1 Registering with the runner...
2 Registered with the runner 77b4c815-ca0a-4001-983e-6c1858bb4bdf
3 [10, 20, 0, 1, 0, 300, 0, 0, 0, 1, 0]
4 [10, 20, 0, 1, -300, 0, 0, 0, 0, 1, 0]
5 [10, 20, 0, 1, 0, -300, 0, 0, 0, 1, 0]
6 [10, 20, 0, 1, 300, 0, 0, 0, 0, 1, 0]
7 [10, 20, 336, 1, 18, 292, 0, 0, 0, 1, 0]
8 [10, 20, 340, 1, -282, -6, 0, 0, 0, 1, 0]
9 [10, 20, 281, 1, 4, -320, 0, 0, 0, 1, 0]

```

Atribut yang diberikan tidak 6, melainkan 11.

Dari paling kiri, informasi yang diberikan adalah sebagai berikut

```

public new List<int> ToStatelist() =>
    new List<int>
    {
        Size,
        Speed,
        CurrentHeading,
        (int) GameObjectType,
        Position.X,
        Position.Y,
        Effects.GetHashCode(),
        TorpedoSalvoCount,
        SupernovaAvailable,
        TeleporterCount,
        ShieldCount
    };
}

```

Dikutip dari [sini](#)

Nah, apabila anda ingin mendapatkan sisa informasi yang sementara ini belum didapatkan, anda bisa memodifikasi file GameObject.java.

Tambahkan sisa atributnya (jangan lupa tambahkan di ctor dan fungsi FromStatelist)

Fungsi fromstatelist nya dibagi jadi 2 kasus.

Game engine dapat memberikan state yang panjangnya 11 (ini object player), assign semuanya ke atribut yang bersesuaian. Tapi bisa jadi game engine ngirimnya ga 11 panjangnya (untuk objek nonplayer, misalnya food kan gadikasih atribut torpedocount), nah itu assign aja sesuai apa yang dikirim trus sisanya assign 0 aja (informasi yang dikirim tetepurut sesuai dari kiri)

[Disini](#) ada enum effects untuk mempermudah membantu decode efeknya. Informasi effect ini sifatnya flag, misal anda menggunakan efek afterburner dan shield secara bersamaan maka value yang dikirim di value effect adalah $1+16 = 17$.

Lalu, setiap constructor GameObject yang ada pada main.java (line 38) bisa ditambah atributnya sesuai dengan yang kalian bikin.

Selamat, anda sudah bisa mengetahui info tambahan terkait efek, torpedo count, supernova, teleporter, dan shield :D

Untuk dapat menjadikan kode kalian menjadi bot yang dapat dipakai, source code kalian harus di-*build* menjadi sebuah file jar terlebih dahulu. Salah satu caranya adalah dengan menggunakan maven ataupun intelliJ.

Apabila menggunakan maven, silahkan menginstall maven terlebih dahulu melalui panduan [berikut](#). Kemudian, untuk build source codenya, silahkan cd ke folder javaBot kemudian memasukkan command berikut di terminal: `mvn clean package`

Setelah beberapa saat, maka **folder target** akan terbentuk yang berisi executable file jar. Silahkan rename file jar tersebut menjadi nama kelompok anda terlebih dahulu. Kemudian, untuk menggunakan file jar tersebut ke permainan, anda dapat mengubah `dotnet ReferenceBot.dll` pada script menjadi `java -jar path` Dengan path adalah path menuju file nama_kelompok.jar hasil build anda.

Setelah anda berhasil build bot Anda, sekarang dapat anda ujicoba dengan melawankan reference bot! Secara default, bot java akan bergerak menuju *food* terdekat sementara bot reference hanya bergerak maju dengan arah tetap. Apabila sudah terlihat perbedaannya, maka **selamat** anda telah berhasil membuat bot galaxio! Silahkan bersenang - senang modifikasi algoritma bot anda hehe

Visualizer

Cara menjalankan permainan melalui visualizer:

1. Lakukan ekstrak pada file zip Galaxio dalam folder “visualiser” sesuai dengan OS kalian
2. Jalankan aplikasi Galaxio
3. Buka menu “Options”
4. Salin path folder “logger-publish” kalian pada “Log Files Location”, lalu “Save”
5. Buka menu “Load”
6. Pilih *file* JSON yang ingin di-load pada “Game Log”, lalu “Start”
7. Setelah masuk ke visualisasinya, kalian dapat melakukan *start*, *pause*, *rewind*, dan *reset*
8. Silahkan buat bot terbaik kalian dan selamat menikmati permainan 😊

Note: Visualizer dapat dilakukan setelah [menjalankan permainan](#)