**بسمه تعالی**

# مستند مسابقه

نبرد هوش مصنوعی ۹۴

بهمن و اسفند ۱۳۹۴

# فهرست مطالب

داستان بازی: چالش یخ و آتش ..... ۲
شرح بازی ..... ۳
نکاتی در مورد بازی و داوری مرحله غیر حضوری ..... ۶
نصب پیش نیازهای لازم ..... ۷
راهنمای رابط برنامه‌نویسی ..... ۸
آغاز ..... ۱۳
نمایشگر گرافیکی بازی ..... ۱۴
نکات مهم ..... ۱۵
ضمیمه۱: شروع به نوشتن کد در Java ..... ۱۶
ضمیمه ۲: شروع به نوشتن کد در C++ ..... ۱۷
ضمیمه ۳: شروع به نوشتن کد در Python ..... ۱۸
ضمیمه ۴: اضافه کردن JDK 1.8 به Eclipse ..... ۱۹

# داستان بازی: چالش یخ و آتش

زمستان در راه است، زمستان در سیاره‌ای خیلی خیلی خیلی دوری به نام «فاپلانیوم[1]».

برخی می گویند که قرار است دست خیلی‌ها کثیف شود، برخی می‌گویند زمستان تنها امید کهکشان است. اما هیچ کس نمی‌داند...

فاپلانیوم همواره از جنگ درون سیاره‌ای رنج می‌برد! جنگی بین قدرت‌مند ترین عناصر کهکشان، یخ و آتش!

تا دورانی که اهالی فاپلانیوم به یاد می‌آورند و در هر دوره‌ای، سیاره درگیر جنگ بین این دو عنصر بود. پیروز این نبرد کنترل سیاره رو به دست می‌گرفت و مغلوب شدگان به غارهایی تاریک و عمیق در نقاط گم‌گشده‌ی سیاره تبعید می‌شدند.

هر موقع که نیروهای آتش پیروز می‌شدند، تابستان فراگیر می‌شد. آب و هوایی خشک و آسمان از دودهای سوزان پر می‌شد. دودهایی که از اقیانوس‌های آتش سطح سیاره بلند می‌شدند.

هر موقع هم که یخ پیروز می‌شد، برف بود که می‌بارید! همه چیز یخ میزد. تنها گرما(!)بخش قلب مرد برفی تصویر دانه‌های زیبای برف بود...

البته محال نیست که بازی برگرده، نیروهای تبعیدی متحد شده و نیروی خود را بازسازی می‌کنند تا در موعد مناسب، جنگی رخ دهد.

بعضی باور دارند که همین جنگ‌ها فاپلانیوم را زنده نگه خواهد داشت و آن را از انقراض حفظ خواهد کرد. چون سیاره تحمل نگه‌داشتن یک عنصر را برای مدت طولانی ندارد و از هم می‌پاشد!

حالا همه چیز عوض شده. نیروهای آتش کنترل سیاره را برای مدتی طولانی به دست گرفته‌اند. برای دهه‌ها، نیروهای آتش چنان قدرت‌مند شده‌اند که یخی‌ها توان مقابله با ارتش سرخ را ندارند. تابستان‌های سوزان فاپلانیوم توازن تمام کهکشان رو به‌هم زده! گفته‌ها حاکی از دوران خاتمه‌ی فاپلانیوم است، اگر اوضاع تغییر نکند...

زمستان در راه است. ارتش یخی‌ها تمام توانش را جمع کرده تا آتشی‌ها را به کام مرگ بفرستند.

و هنوز تا تغییر راه زیادی مانده...

[1] در زبان فالانگیم، «فاپلانیوم» به معنای سیاره دور است.

# شرح بازی

**بازی:** یک بازی دو نفره نوبتی است که هر بازیکن باید با حرکت‌دادن غول‌های خود روی یک گراف، سعی در تصاحب رأس‌ها و غول‌های بیشتر داشته باشد تا به پیروزی برسد.

**نقشه بازی:** نقشه بازی یک گراف است که هر رأس آن یا خالی است، یا دارای غول بازیکن اول، یا غول بازیکن دوم. مالکیت رأس‌های دارای غول، معادل صاحب غول است و مالک رأس‌های خالی می‌تواند بازیکن اول یا بازیکن دوم یا هیچ‌کدام باشد. به هر غول یک عدد صحیح به عنوان قدرت آن نسبت داده می‌شود.

**شروع بازی:** در شروع غول‌های دو بازیکن به صورت کاملاً قرینه روی گراف پخش شده‌اند و مالکیت رأس‌ها نیز کاملاً قرینه است.

**روند بازی:** در هر نوبت ابتدا بازیکن‌ها فعّالیت می‌کنند و سپس خود بازی تغییراتی را روی غول‌ها اعمال می‌کند.

**فعالیت بازیکن:** هر بازیکن در هر نوبت به ازای هر غول خودش این حق انتخاب را دارد که قسمتی (مقداری صحیح) از آن یا تمام آن غول را به یکی از رأس‌های مجاور بفرستد. (آن غول از نظر قدرت تقسیم می‌شود.)

**پیامد فعالیت بازیکن:**

فعالیت بازیکنان در هر نوبت به صورت همزمان است. به این معنی که فعالیت دو بازیکن از نظر زمانی تقدمی به یکدیگر ندارد. البته اتفاقاتی که در هر نوبت در بازی می افتد، ترتیب مشخصی دارد.

۱- اگر غول بازیکن اول و دوم حین حرکت روی یال باهم برخورد کردند، طبق قوانین درگیری که بعداً گفته خواهد شد هر دو نابود شده یا یکی از آن‌ها با دادن تلفات پیروز می‌شود و به مسیر خود ادامه می‌دهد.

۲- مالکیت رأس‌هایی که بعد از انجام جابه‌جایی‌ها، روی آنها هیچ غولی وجود ندارد، عوض نمی‌شود.

۳- اگر در یک رأس فقط غول‌های یک بازیکن وجود داشته باشند، آن غول‌ها با هم یکی می‌شوند. (قدرت غول بوجود آمده، برابر با مجموع قدرت غول‌ها است.) و مالکیت آن رأس با مالک غول‌ها است.

۴- روی رأس‌هایی که غول‌های دو بازیکن وجود دارند غول‌های هر کدام از بازیکنان مانند قسمت ۳ با هم متحد می‌شوند و سپس طبق قوانین باهم درگیر می‌شوند. اگر غولی از جنگ باقی ماند، مالکیت رأس مشابه آن می‌شود و اگر همه‌ی غول‌ها از بین رفتند، مالکیت آن راس به همان صورت قبل از درگیری باقی می ماند.

۵- به ازای هر رأس، غولی که در جنگ روی آن رأس شکست میخورد یعنی غولی که قدرت کمتری دارد ابتدا تقسیم شده، و از طریق یال‌ها به رأس‌های مجاور که صاحب آن مشابه غول شکست خورده است، فرار انجام می شود. سپس جنگ اتفاق می افتد و قدرت غول باقیمانده ی بازنده پس از فرار صفر می شود. قدرت غول فراری از هر یال مقداری بیشینه دارد (این مقدار صحیح برای همه ی یال ها در یک نقشه یکسان است.) و طبیعتاً مقداری از غول شکست خورده ممکن است باقی بماند و نتواند فرار کند که در این صورت قدرت غول

باقیمانده در جنگ صفر می شود. در صورتی که مجموع مقدار فرار ممکن از یال ها (که سر دیگر آن ها راس هایی با مالکیت مشابه با غول شکست خورده است) بیشتر از قدرت غول شکست خورده باشد به صورت تصادفی به تعدادی از راس های ممکن (یا همه ی آن ها) فرار می کند و چیزی از آن غول در آن راس باقی نمی ماند. به طور مثال اگر غولی با قدرت چهار روی راسی باشد که سه یال به راس هایی با مالکیت یکسان دارد و بیشینه مقدار فرار ممکن از هر یال برابر دو است، آنگاه در هنگام جنگ این غول به دو قسمت با قدرت دو تقسیم شده و به صورت تصادفی به یکی از سه راس ممکن فرار می کند و راسی که قبلا در آن بوده به طور کامل خالی می شود.

**اعمال بازی پس از فعالیت بازیکن‌ها:** اگر غولی مالکیت یک رأس را در این نوبت گرفته باشد و در نوبت قبل مالکیت این رأس متفاوت بوده باشد، قدرت آن غول به میزان مشخص افزایش می‌یابد. همچنین به ازای هر یال گراف که مالک دو راس سر آن یک بازیکن است قدرت هر کدام از این غول‌ها به میزان مشخص افزایش می‌یابد. و اگر غولی در یکی از آن رئوس وجود نداشته باشه، غولی با آن قدرت مشخص در آن راس بوجود می آید. افزایش به دلیل اینکه مالکیت دو سر یال برای یک بازیکن است در هر نوبتی که چنین شرایطی وجود داشته باشد اتفاق می افتد. ولی افزایش به دلیل به دست آوردن مالکیت تنها در نوبتی که مالکیت یک راس عوض شود، اتفاق می افتد.

ترتیب انجام اتفاقات بازی به این صورت است: حرکت ها و جنگ های روی یال، فرار، جنگ های روی راس، اضافه شدن نیرو ها به علت گرفتن مالکیت و یا یال هایی مالک دو سر آن ها یکسان است.

هر کدام از مجموعه اتفاقاتی که گفته شد، در یک نوبت بازی به صورت همزمان انجام می شود. به طور مثال تمامی فرار ها به طور همزمان انجام می شوند.

**شرایط پایان بازی:** اگر یک بازیکن هیچ غولی نداشته باشد یا تعداد نوبت به حداکثر برسد، بازی پایان می‌یابد و بازیکنی که مجموع قدرت غول‌هایش بیشتر باشد برنده می‌شود. حداکثر تعداد نوبت ها برای هر نقشه متفاوت است و با استفاده از رابط برنامه نویسی قابل دستیابی است. امتیاز بازیکنان پس از اتمام بازی به این صورت محاسبه می شود:

$$\text{اگر بازیکن برنده باشد : امتیاز} = \frac{\text{نوبت های باقیمانده}}{\text{حداکثر نوبت ها}} + \frac{\text{مجموع قدرت غولهای برنده}}{\text{مجموع قدرت کل غولها}} + ۱$$

$$\text{اگر بازیکن بازنده باشد : امتیاز} = \frac{\text{نوبت های باقیمانده}}{\text{حداکثر نوبت ها}} - \frac{\text{مجموع قدرت غولهای بازنده}}{\text{مجموع قدرت کل غولها}} + ۱$$

**اطلاعات بازیکن‌ها:** هر بازیکن به طور کامل از ساختار نقشه و مالکیت رأس‌ها و غول‌های خودش آگاهی دارد. از مکان غول‌های تیم مقابل نیز آگاه است امّا مقدار آن ها را به صورت تقریبی در سه سطح: قوی – متوسط – ضعیف می‌بیند.

### قوانین درگیری:

غولی در جنگ پیروز می شود که قدرت آن بیشتر است و غول باقیمانده از بازنده بعد از فرار به طور کامل نابود می شود. البته غول برنده نیز در جنگ مقداری تلفات می دهد. این تلفات با توجه به سطح تقریبی غول بازنده پس از فرار و ضریبی از قدرت غول بازنده پس از فرار است که به طور کامل نابود می شود. در صورتی که قدرت دو غول حریف در جنگ برابر باشد، هیچ فراری اتفاق نمی افتد و هر دو به طور کامل نابود می شوند. توجه کنید که فرار فقط برای جنگ های روی راس است و برای جنگ های روی یال فرار وجود ندارد.

۱- اگر سطح تقریبی (ضعیف، متوسط، قوی) دو غول یکسان باشد، به مقدار قدرت غول نابود شده بازنده، از قدرت برنده کم می‌شود.

۲- اگر سطح تقریبی برنده قوی و سطح تقریبی بازنده متوسط باشد به مقدار سقف $\frac{2}{3}$ قدرت غول نابود شده بازنده، از قدرت برنده کم می‌شود. در صورتی که قدرت برنده متوسط و قدرت بازنده ضعیف باشد، هم همین اتفاق می افتد. در این زمان می گوییم که سطح تقریبی برنده و بازنده اختلافی برابر یک دارد.

۳- اگر سطح تقریبی برنده قوی و سطح تقریبی بازنده ضعیف باشد به مقدار سقف $\frac{1}{3}$ قدرت غول نابود شده بازنده، از قدرت برنده کم می‌شود. در این زمان می گوییم که سطح تقریبی برنده و بازنده، اختلافی برابر دو دارد.

ثابت های بازی در جدول زیر آورده شده است.

| نام ثابت | مقدار ثابت |
|---|---|
| **بیشینه مقدار فرار از هر یال** | ۲ |
| **مقدار افزایش قدرتی که به خاطر به دست آوردن مالکیت یک راس اتفاق می افتد** | ۴ |
| **مقدار افزایش قدرتی که به علت وجود یال خودی اتفاق می افتد.** | ۱ |
| **مقدار بیشینه قدرت ممکن برای یک غول ضعیف** | ۱۰ |
| **مقدار بیشینه قدرت ممکن برای یک غول متوسط** | ۳۰ |
| **ضریب تلفات برنده نسبت به بازنده جنگ زمانی که اختلاف تقریبی آن ها یکی است.** | $\frac{۲}{۳}$ |
| **ضریب تلفات برنده نسبت به بازنده جنگ زمانی که اختلاف تقریبی آن ها دو است.** | $\frac{۱}{۳}$ |

## نکاتی در مورد بازی و داوری مرحله غیر حضوری

در این قسمت توضیحاتی در مورد بازی و رقابت های آن داده می شود.

شما باید در نقش یک بازیکن بازی که توضیح داده شد، یک برنامه ی هوش مصنوعی بنویسید که غول هایتان را کنترل کند و سعی در بردن از حریف خود کنید.

**توجه:** در مسابقه ی امسال بازی مرحله ی حضوری و غیر حضوری کاملا یکسان است ولی ممکن است که ثابت های بازی در مرحله ی حضوری تغییر کند.

### نکاتی در مورد بازی

- حداکثر تعداد رئوس نقشه ۱۰۰ است.
- برای اجرای بازی ها حتما باید دو client اجرا شوند. برای client دوم یک فایل jar به شما داده شده است. توجه کنید که این client صرفا الگوریتم تصادفی پیش فرض است.
- هر بازی بین دو تیم برگزار خواهد شد.
- نقشه های مورد استفاده به جهت داوری برای تمام تیم ها یکسان هستند.

### نکاتی در مورد محدودیت های اجرا

- قدرت پردازشی که در اختیار کد شما قرار می گیرد 2GH است.
- میزان حافظه ای که در اختیار کد شما قرار می گیرد 1GB است.

# نصب پیش نیازهای لازم

برای اجرای بازی و نوشتن کدهای خود نیاز به نصب پیش نیازهای زیر دارید.

1. Java Development Kit 1.8.0
2. Python 3
3. C++ 11

مورد ۱ برای اجرای بازی ضروری است. مورد ۲ فقط در صورتی لازم است بخواهید با Python کد بزنید و مورد ۳ فقط در صورتی لازم است که بخواهید با C++ کد بزنید. توجه کنید که از زبان C++ فقط تحت سیستم عامل لینوکس پشتیبانی می‌شود. در ادامه نحوه نصب هر کدام از این پیش نیازها توضیح داده خواهد شد.

## ۱) Java Development Kit 1.8.0

برای دریافت این پیش نیاز می‌توانید به لینک زیر مراجعه کنید و فایل سازگار با سیستم عامل خود را دریافت کنید.

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

برای دیدن راهنمای نصب این پیش نیاز در سیستم عامل‌های مختلف می‌توانید به لینک‌های زیر مراجعه کنید.

| | |
|---|---|
| Windows | http://www.wikihow.com/Install-the-Java-Software-Development-Kit |
| Linux | http://www.wikihow.com/Install-Oracle-Java-JDK-on-Ubuntu-Linux |
| Mac-OS | http://www.wikihow.com/Install-the-JDK-(Java-Development-Kit)-on-Mac-OS-X |

## ۲) Python 3

نصب این پیش نیاز برای گروه‌هایی لازم است که قصد استفاده از Client پایتون را دارند.

برای دریافت این پیش نیاز، با توجه به سیستم عامل خود به یکی از لینک‌های زیر مراجعه کنید.

| | |
|---|---|
| Windows | https://www.python.org/downloads/windows/ |
| Linux | https://www.python.org/downloads/source/ |
| Mac-OS | https://www.python.org/downloads/mac-osx/ |

برای دیدن راهنمای نصب این پیش نیاز می‌توانید به لینک زیر مراجعه کنید.

www.wikihow.com/Install-Python

## ۳) C++11 Compiler

گروه‌هایی که قصد استفاده از Client زبان C++ را دارند، باید یک کامپایلر با قابلیت کامپایل کردن کدهای C++11 را داشته باشند.

Client این زبان روی سیستم عامل لینوکس کامپایل می‌شود.

# راهنمای رابط برنامه‌نویسی

کلاسی به نام AI در اختیار شما قرار داده شده‌است. در این کلاس تابعی به نام doTurn وجود دارد. در هر نوبت، این تابع یک بار صدا زده می‌شود و شما باید در این تابع دستوراتی که می‌خواهید را به غول‌های خود بدهید. پس شما باید این تابع را به گونه‌ای بنویسید که غول‌هایتان هوشمندانه عمل کنند. هر نوبت بازی یک ثانیه طول می‌کشد که شما در 500 میلی ثانیه اوّل آن فرصت دارید تا دستورات لازم را به غول‌هایتان بدهید. تابع doTurn به عنوان ورودی یک شیء از نوع رابط World می‌گیرد. این شیء بیانگر حالت کنونی جهان است و درون آن، تمامی اطّلاعاتی قرار دارد که شما به آن‌ها دسترسی دارید. در زیر شکل کلّی این کلاس در زبان‌های متفاوت را مشاهده می‌کنید. کسانی که از Python استفاده می‌کنند توجه کنند که تمام مواردی که لازم دارند در فایل Model.py است و اطّلاعات آن کافی است. برای اطّلاعات بیشتر می توانید به مطالب زیر مراجعه کنید.

**توجه:** در صورتی که تفاوت های جزئی بین توضیحات این بخش و کد کلاینتی که در اختیار شما قرار گرفته وجود داشت، معیار کد مربوطه است.

Python
```
class AI():
  def do_turn(world):
    // AI code comes here
```

C++
```
class AI {
  void doTurn(World*
world) {
    // AI code comes here
  }
};
```

Java
```
class AI {
  doTurn(World world) {
    // AI code comes here
  }
}
```

## رابط World

این رابط (در Python یک کلاس است) شامل اطلاعاتی از شرایط حال حاضر بازی است. دقت کنید که در هر نوبت این اطلاعات تغییر می‌کنند و اگر شما می‌خواهید که به اطلاعات گذشته دسترسی داشته باشید باید خودتان آن را ذخیره کنید. نحوه دسترسی به اطلاعاتی از این کلاس که می‌توانید استفاده کنید را در ادامه توضیح می‌دهیم.

**My Id:**

دسترسی به شماره‌ی تیم خود. (شماره‌ی هر تیم، یک یا صفر است.)

Python
```
World.my_id
```

C++
```
int getMyId()
```

Java
```
int getMyId()
```

**Map:**

دسترسی به نقشه (گراف) بازی و اطلاعات آن.

Python
```
World.map
```

C++
```
Graph* getMap()
```

Java
```
Graph getMap()
```

**My Nodes:**

دسترسی به رئوسی که بازیکن مالک آن است.

| Python | C++ | Java |
|---|---|---|
| `World.my_nodes` | `std::vector<Node*>& getMyNodes()` | `Node[] getMyNodes()` |

**Opponent Nodes:**

دسترسی به رئوسی که حریف مالک آن است.

| Python | C++ | Java |
|---|---|---|
| `World.opponent_nodes` | `std::vector<Node*>& getOpponentNodes()` | `Node[] getOpponentNodes()` |

**Free Nodes:**

دسترسی به رئوسی که بدون مالک هستند.

| Python | C++ | Java |
|---|---|---|
| `World.free_nodes` | `std::vector<Node*>& getFreeNodes()` | `Node[] getFreeNodes()` |

**Total Turn:**

دسترسی به تعداد کل نوبت های بازی.

| Python | C++ | Java |
|---|---|---|
| `World.total_turn` | `int getTotalTurn()` | `int getTotalTurn()` |

**Turn Number:**

دسترسی به شماره‌ی نوبت کنونی بازی. (با شروع از صفر)

| Python | C++ | Java |
|---|---|---|
| `World.turn_number` | `int getTurnNumber()` | `int getTurnNumber()` |

**Total Turn Time:**

دسترسی به محدودیت زمانی برای مدت پردازشی که بازیکن در یک نوبت می تواند انجام دهد. (بر حسب میلی ثانیه)

| Python | C++ | Java |
|---|---|---|
| `World.total_turn_time` | `long long getTotalTurnTime()` | `long getTotalTurnTime()` |

**Turn Time Passed:**

دسترسی به مدت زمانی که برای پردازش نوبت فعلی سپری شده‌است. (بر حسب میلی ثانیه)

| Python | C++ | Java |
|---|---|---|
| `World.get_turn_time_passed()` | `long long getTurnTimePassed()` | `long getTurnTimePassed()` |

**Turn Remaining Time:**

دسترسی به مدت زمانی که برای پردازش نوبت فعلی باقی مانده‌است. (بر حسب میلی ثانیه)

| Python | C++ | Java |
|---|---|---|
| `World.get_turn_remaining_time()` | `long long getTurnRemainingTime()` | `long getTurnRemainingTime()` |

**Escape Constant:**

دسترسی به مقدار ثابتی که می تواند از هر یال فرار کند.

| Python | C++ | Java |
|---|---|---|
| `World.escape` | `int getEscapeConstant()` | `int getEscapeConstant()` |

**Node Bonus Constant:**

دسترسی به مقدار ثابتی که به یک غول بابت به دست آوردن مالکیت یک خانه، اضافه می شود.

| Python | C++ | Java |
|---|---|---|
| `World.node_bonus` | `int getNodeBonusConstant()` | `int getNodeBonusConstant()` |

**Edge Bonus Constant:**

دسترسی به مقدار ثابتی که به یک غول بابت یالی که مالک سر دیگرش خودش است اضافه می شود.

| Python | C++ | Java |
|---|---|---|
| `World.edge_bonus` | `int getEdgeBonusConstant()` | `int getEdgeBonusConstant()` |

**Low Army Max Constant:**

دسترسی به مقدار قدرت بیشینه ممکن برای یک غول ضعیف. قدرت هر غول می تواند به هر اندازه ی افزایش یابد ولی اگر قدرت آن بیش از این مقدار شود، دیگر آن غول یک غول ضعیف نیست و ممکن است یک غول متوسط یا قوی باشد.

| Python | C++ | Java |
|---|---|---|
| `World.low_army_bound` | `int getLowArmyBound()` | `int getLowArmyBound()` |

**Medium Army Max Constant:**

دسترسی به مقدار قدرت بیشنیه ممکن برای یک غول متوسط. قدرت هر غول می تواند به هر اندازه ی افزایش یابد ولی اگر قدرت آن بیش از این مقدار شود، دیگر آن غول یک غول متوسط یا ضعیف نیست و یک غول قوی است.

| Python | C++ | Java |
|---|---|---|
| `World.medium_army_bound` | `int getMediumArmyBound()` | `int getMediumArmyBound()` |

**Medium Casualty Constant:**

دسترسی به ضریب تلفات برنده نسبت به بازنده جنگ در زمانی که اختلاف تقریبی آن ها به اندازه ی یکی است. به طور مثال برنده قوی و بازنده متوسط است و یا اینکه برنده متوسط و بازنده ضعیف است.

| Python | C++ | Java |
|---|---|---|
| `World.medium_casualty_coefficient` | `double getMediumCasualtyCoefficient()` | `double getMediumCasualtyCoefficient()` |

**Low Casualty Constant:**

دسترسی به ضریب تلفات برنده نسبت به بازنده جنگ در زمانی که اختلاف تقریبی آن ها به اندازه ی دو است. به طور مثال برنده قوی و بازنده ضعیف است.

| Python | C++ | Java |
|---|---|---|
| `World.low_casualty_coefficient` | `double getLowCasualtyCoefficient()` | `double getLowCasualtyCoefficient()` |

**Move Army:**

تقسیم قدرت غول رأس src و انتقال غولی با قدرت count به رأس dst. در اینجا src و dst از نوع int هستند و اندیس رئوس را مشخص می کنند. در صورتی که برای غول یک راس (src) چندین بار تابع moveArmy صدا زده شود، فقط آخرین حرکت مجاز برای آن راس انجام می شود و بقیه ی حرکات در نظر گرفته نمی شود. منظور از حرکت مجاز حرکتی است که قابل انجام باشد، با توجه به قدرت غول آن راس و یال های گراف.

| Python | C++ | Java |
|---|---|---|
| `World.move_army(src, dst, count)` | `void moveArmy(int src, int dst, int count);` | `void moveArmy(int src, int dst, int count);` |

**Move Army:**

تقسیم قدرت غول رأس src و انتقال غولی با قدرت count به رأس dst. در اینجا src و dst از نوع Node هستند.

| Python | C++ | Java |
|---|---|---|

```
World.move_army(src, dst, count)
```

```
void moveArmy(Node* src, Node* dst, int count);
```

```
void moveArmy(Node src, Node dst, int count);
```

## کلاس Graph

رئوس گراف و توابع دسترسی به آن‌ها در این کلاس وجود دارد.

**Nodes:**

دسترسی به رئوس گراف.

Python
```
Graph.nodes
```

C++
```
std::vector<Node*>& getNodes()
```

Java
```
Node[] getNodes()
```

**Node:**

دسترسی به یک رأس گراف با استفاده از اندیس آن.

Python
```
Graph.nodes[index]
```

C++
```
Node* getNode(int index)
```

Java
```
Node getNode(int index)
```

## کلاس Node

این کلاس شامل اطلاعات و توابع مربوط به رأس است.

**Neighbours:**

دسترسی به رئوس مجاور (همسایه).

Python
```
Node.neighbours
```

C++
```
std::vector<Node*>& getNeighbours()
```

Java
```
Node[] getNeighbours()
```

**Index:**

دسترسی به اندیس رأس.

Python
```
Node.index
```

C++
```
int getIndex()
```

Java
```
int getIndex()
```

**Owner:**

دسترسی به شماره‌ی مالک رأس. شماره‌ی بازیکنان یا صفر است و یا یک. این تابع برای رأس های بدون مالک منفی یک برمی گرداند.

Python

C++

Java

| Python | C++ | Java |
|---|---|---|
| `Node.owner` | `int getOwner()` | `int getOwner()` |

**Army Count:**

دسترسی به قدرت غولی که روی رأس قرار گرفته است. اگر مالک این رأس همان بازیکنی باشد که این تابع را صدا زده است مقدار دقیق به او برگردانده می شود و در صورتی که مالک، حریف او باشد، مقدار تقریبی به او برگردانده می شود. برای مقادیر تقریبی مقادیر صفر، یک و دو به ترتیب برای غول ضعیف، متوسط و قوی برگردانده می شود.

| Python | C++ | Java |
|---|---|---|
| `Node.army_count` | `int getArmyCount()` | `int getArmyCount()` |

# آغاز

برای اجرای بازی باید یک سرور اجرا و دو کلاینت اجرا و به آن متصل شوند. این کلاینت ها میتوانند از هر زبانی (Java، C++ و Python) باشند. برای اجرای سرور کافی است فایل FlowsGameServer.jar را اجرا نمایید. بعد از اجرا باید نقشه ی بازی را انتخاب نمایید. در مرحله ی بعد سرور منتظر اتصال کلاینت ها خواهد ماند. شما میتوانید کلاینت خود را به گونه ای که در ضمیمه های ۱ تا ۳ توضیح داده شده است اجرا نمایید و بعد از اتصال آن به سرور با یک کلاینت دیگر یا حتی با کد خودتان بازی کنید. همچنین میتوانید از روبات تصادفی داده شده به عنوان حریف تمرینی استفاده نمایید. در صورتی که سوالی داشتید یا اشکالی در کارکرد سرور یا کلاینت ها مشاهده نمودید، بعد از اطمینان از صحت کد خود از طریق سامانه پرسش و پاسخ ما را در جریان قرار دهید.

**توجه:** در صورتی که می خواهید در هر بار اجرای بازی آدرس نقشه را وارد نکنید می توانید آدرس نقشه را در فایل پیکربندی (config) سرور وارد کنید. برای این کار ابتدا با دستور زیر یک فایل پیکربندی تولید کنید.

```
java -jar FlowsGameServer-v1.0.jar --generate-config=game.conf
```

سپس این فایل را با یک ویرایشگر متن باز نموده و روبروی فیلد آدرس مطلق (absolute)، آدرس فایل نقشه را قرار دهید و فایل پیکربندی را ذخیره کنید. از این پس برای اجرای سرور از دستور زیر استفاده نمایید.

```
java -jar FlowsGameServer-v1.0.jar --config=game.conf
```

سرور نقشه را تشخیص خواهد داد و دیگر از شما محل نقشه را نخواهد پرسید.

## نمایشگر گرافیکی بازی

راس‌های شکل، نشان دهنده‌ی جایگاه‌های نیروهای بازی است و یال‌های بین راس‌ها نشان دهنده‌ی مسیر بین آن‌هاست و در هر مرحله انتقال نیروها از یک راس به راس دیگر با یک فلش روی یال بین دو راس در جهت انتقال نیروها نمایش داده می‌شود.

فلش، هم‌رنگ با راسی نمایش داده می‌شود که نیروهایش را منتقل می‌کند.

تعداد نیروهای منتقل شده نیز توسط عددی روی فلش نمایش داده شده است. در صورتی که عملیات انتقال مجاز باشد دور عددی که نشان دهنده‌ی تعداد نیروها است، دایره‌ای کشیده می‌شود.

عدد وسط و بالای پنجره نمایش دهنده‌ی تعداد مراحل رد شده از بازی است.

عدد بالا سمت چپ پنجره، نمایش دهنده‌ی امتیازات نفر اول بازی است و عدد مقابل آن، امتیازات نفر دوم.

با استفاده از دستورات **change theme**، تم رنگ زمینه و راس‌ها تغییر می‌کند.

با استفاده از دستور **pause game**، بازی را متوقف می‌کنید و با دستور **next step** مرحله‌ی بعد نمایش داده می‌شود.

دستور **start recording**، از تصویر صفحه‌ی هر مرحله‌ی بازی را ذخیره می‌کند.

**توجه:** این نمایشگر صرفا برای اجرای **local** بازی به صورت ساده است.

## نکات مهم

- برای هر تیم آخرین کدی که بارگذاری شده است، برای اجرا استفاده می شود. این کد می تواند به هر کدام از سه زبان Java، C++ و Python باشد.
- کلاس‌ها و توابعی که در طول مستند بررسی شدند بیشتر مورد نیاز هستند، ولی برای آشنایی بیشتر با نحوه عملکرد می‌توانید به خود کد مراجعه کنید.
- نحوه‌ی ارسال پاسخ‌ها متعاقبا اعلام خواهد شد.

# ضمیمه۱: شروع به نوشتن کد در Java

## طریقه کد نویسی

- شما باید کد هوش مصنوعی خود را در تابع doTurn در فایل AI.java قرار دهید.
- شما مجاز هستید که همه ی فایلهای موجود در پکیج client را تغییر دهید ولی ما توصیه می کنیم که به جز AI.java فایل دیگری را تغییر ندهید. مسئولیت عواقب هر گونه تغییر دیگر بر عهده ی شرکت کننده است.

## نحوه اجرا

۱. در صورتی که JDK 1.8 را نصب کرده اید ولی در eclipse خود به آن دسترسی ندارید به ضمیمه ۴ مراجعه کنید.

۲. فولدر AIC16-Client-Java را داخل IDE خود، به عنوان مثال Eclipse، Import کنید. برای این کار به منوی File رجوع کرده و گزینه Import را انتخاب کنید. سپس در پنجره باز شده در پوشه General گزینه ی Existing Projects into Workspaces را انتخاب کنید و سپس Next را کلیک کنید. در صفحه ی جدید گزینه ی Select archive file را انتخاب کنید و آدرس فایل AIC16-Client-Java.zip را وارد کنید. سپس بر روی Finish کلیک کنید. حال باید libraryهای استفاده شده را به پروژه اضافه کنیم. برای این کار در قسمت Package Explorer (سمت چپ eclipse) در فولدر libs روی فایل gson-2.3.1.jar راست کلیک کرده و در زیرمنوی Build Path، گزینه ی Add to Build Path را انتخاب نمایید.

۳. پس از Import کردن این فولدر، یک بار آن را Run کنید تا از Build شدن آن اطمینان حاصل کنید. در صورتی که درست Build شود، ممکن است Exception ای با محتوای Cannot connect to server مشاهده کنید که به علت در حال اجرا نبودن سرور است.

## ارسال فایل ها

برای ارسال فایل های خود، پوشه src را zip کنید و این فایل فشرده را ارسال کنید.

# ضمیمه ۲: شروع به نوشتن کد در C++

## طریقه کد نویسی

- شما باید کد هوش مصنوعی خود را در تابع doTurn در فایل AI.cpp قرار دهید.
- شما می توانید همه ی فایل ها را تغییر دهید؛ اما توصیه می کنیم فقط فایل AI.cpp را تغییر دهید. مسئولیت عواقب هر گونه تغییر دیگر بر عهده ی شرکت کننده است.
- شما مجاز هستید هر تعداد فایل که می خواهید به فایل های کلاینت اضافه نمایید، با این شرط که آن ها را به make file پروژه اضافه نمایید.

## نحوه اجرا

برای نصب client مربوط به C++ در Linux ابتدا باید packageهای زیر را با استفاده از وارد کردن آن‌ها در terminal نصب کنید.

Compiler C++11: sudo apt-get install build-essential

برای اجرای پروژه Client C++ باید فایل zip ضمیمه شده به نام AIC16-Client-cpp.zip را استخراج کنید و دستور make را در آن پوشه اجرا کنید.

ممکن است Exception ای با محتوای Connection Refused یا Error While Connecting to Server مشاهده کنید که به علت در حال اجرا نبودن سرور است.

کاربرانی که از زبان C++ استفاده می‌کنند در صورت لزوم، برای فرستادن کد خود در جاج مسابقات باید فایل Makefileی موجود را تغییر دهند. نام فایل خروجی که Makefile شما تولید می کند باید flows.out باشد.

## ارسال فایل ها

برای ارسال فایل های خود، تمامی فایل های مربوط به client را به همراه Makefile خود zip کنید و این فایل را ارسال کنید.

# ضمیمه ۳: شروع به نوشتن کد در Python

## طریقه کد نویسی

- شما باید کد هوش مصنوعی خود را در تابع do_turn در فایل AI.py قرار دهید.
- شما می توانید همه ی فایل ها را تغییر دهید؛ اما توصیه می کنیم فقط فایل AI.py را تغییر دهید. مسئولیت عواقب هر گونه تغییر دیگر بر عهده ی شرکت کننده است.
- شما مجاز هستید هر تعداد فایل که می خواهید به فایل های کلاینت اضافه نمایید.

## نحوه اجرا

در ابتدا با استفاده از دستور العمل گفته شده در ضمیمه ۱، سرور را اجرا کنید و سپس با توجه به نوع سیستم عامل خود به یکی از روش‌های زیر client خود را اجرا کنید.

- Windows: فایل Contoller.py را اجرا کنید. در صورتی که با اجرای Controller.py مشکل دارید و Python 2 را از قبل نصب داشته اید، می‌توانید با کلیک راست روی فایل Controller.py به قسمت open with بروید و با انتخاب python.exe از مسیر نصب Python 3 کد را اجرا کنید.
- Linux: ترمینال را باز کرده و وارد پوشه ی AIC16-Client-Python بشوید. سپس با اجرای دستور زیر کد را اجرا کنید.

```
python3 Controller.py
```

منتظر بمانید تا پیام connected to server چاپ شود. اگر پیام خطا چاپ شد، اطمینان حاصل کنید که سرور در حال اجرا است.

## ارسال فایل ها

برای ارسال فایل های خود، تمامی فایل های مربوط به client را zip کنید و این فایل را ارسال کنید.

## ضمیمه ۴: اضافه کردن JDK 1.8 به Eclipse

توجه کنید که اگر از eclipse نسخه[۲] Luna استفاده می‌کنید به این راهنما احتیاجی ندارید، ولی اگر از نسخه Kepler استفاده می کنید به راهنمایی زیر مراجعه کنید. نسخه‌های دیگر eclipse از JDK 1.8 پشتیبانی نمی کنند و اگر نسخه قدیمی تری دارید باید یکی از این دو نسخه را دانلود کنید.

ابتدا از نوار بالایی گزینه ی help را انتخاب کرده و سپس زیر منوی Eclipse Marketplace را انتخاب کنید. در فیلد Find عبارت java 8 را وارد کنید و java 8 support for Eclipse.. را نصب کنید. سپس eclipse خود را restart کنید.

بر روی پروژه کلیک راست کنید و گزینه ی properties را انتخاب کنید. سپس از منوی سمت چپ java compiler را انتخاب کنید و compliance level آن را به ۱٫۸ تغییر دهید.

حال بر روی java build path از منوی سمت چپ properties کلیک کنید و در صورتی که jre 1.8 دچار مشکل بود روی آن کلیک کنید و دکمه ی Edit را بزنید و دکمه ی installed JREs را انتخاب کنید. در صورتی که jre 1.8 وجود نداشت دکمه ی add را بزنید و پس از انتخاب Standard VM دکمه ی next را زده و پس از آن آدرس jre 1.8 خود را با زدن دکمه ی Directory وارد کنید. سپس بر روی Finish کلیک کنید.

[2] version