The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2016
CSSE2310 / CSSE7231 - Assignment 4


# 1 Introduction

In this assignment, you will write two `C99` programs to carry out the "agent" simulation described below. The first program (`2310team`) will be used to represent a team of agents in the simulation. The second program (`2310controller`) will control the simulation environment. There will be no user input in this assignment. Instead, the configuration for the simulation will be read from files.

*Your programs must not create any files on disk not mentioned in this specification. Your programs must not execute other programs.*

# 2 Context

The simulation will involve teams battling each other through the use of agents and their attacks.

A team will have four agents which will battle against the four agents of an opposing team. An agent has a type and a set of attacks it can use. Each attack, which also has a type, will have a different level of effectiveness (high, normal or low) against an opponent agent, depending on the types of the both the attack and the opponent. Agents will take damage from an attack based on its effectiveness. Once an agent has lost 10 health, it is eliminated and the next agent in the team will take its place in the battle. The battle is over once all agents in one team have been eliminated.

Teams can also participate in a larger simulation with multiple teams, through a controller. The controller will wait for a number of teams to join, then play out one or more rounds. In each round, a team will battle all the other teams in its zone, then report back to the controller. After all of the battles in a round are complete, teams will move to new zones to battle the teams there. This continues until all of the rounds are complete.

# 3 Team

## 3.1 Invocation / Modes

The `2310team` program will have three modes of operation. The first mode ("simulation") will connect to a controller process to be part of a larger simulation. The second("wait") and third("challenge") forms are used for a single battle between two teams (i.e. there is no controller involved). In this case the description of the battle will be printed by both teams and then they will both exit.

### 3.1.1 Simulation

Usage: `2310team` *`controllerport teamfile`*

This mode will connect the team to a `2310controller` process so it can be part of a simulation. As soon as the team connects, it will receive the sinister information from the controller. After this, it will load its team and start waiting on an OS assigned port for challenges from other teams. The team then lets the controller know that it would like to be involved in the simulation by sending an `iwannaplay` message.

Following this setup, the team will follow the sequence of operations outlined in Section 4.2.

### 3.1.2 Wait

Usage: `2310team wait` *`teamfile sinisterfile`*

The `wait` process will listen on an OS chosen port for a `challenge` team. Upon starting, the process will print out the port it is listening on. For example:

```
57829
```

After a `challenge` team connects to the port, both teams will engage in a battle, followed by the narrative being printed to `stdout`.

### 3.1.3 Challenge

Usage: `2310team challenge` *`teamfile sinisterfile targetport`*

The `challenge` team will connect to the `wait` team on the specified target port. A battle will then play out between the two teams, followed by the narrative being printed to `stdout`.

## 3.2 Sequence

The two teams will start by setting up ready for a battle:

1. Challengers send team name using `fightmeirl`
2. Waiting team responds using `haveatyou`
3. Challengers select first agent and notify the waiting team
4. Waiting team responds with their first agent

Once the setup is complete, the battle can commence. The challengers will attack first. The battle will consist of attacks made in the following order:

1. Team 1 attacks using its current agent
2. Team 2 takes damage from the attack
3. Team 2 attacks using its current agent
4. Team 1 takes damage from the attack
5. Repeat, until all agents of one team are eliminated

Each agent in a team starts with 10 points of health. An attack will remove a number of points of health from the targeted agent, as indicated in Table 1. When a team's current agent is eliminated, it is replaced by the next agent and the opposition is sent an `iselectyou` message.

| **Effectiveness** | **Points lost** |
|---|---|
| High | 3 |
| Normal | 2 |
| Low | 1 |

Table 1: Damage taken from attacks

When all agents of one team have been eliminated, the battle is over. Both teams will then print out a narrative for the battle, in the following format:
*otherteam* has a difference of opinion
*challengeteam* chooses *agent*
*waitteam* chooses *agent*
*agent* uses *attack*: *effectiveness string*
*agent* uses *attack*: *effectiveness string*
...
*agent* uses *attack*: *effectiveness string* - *agent* was eliminated.
...
Team *teamname* was eliminated.

Note:

- In `challenge` and `wait` modes, only the messages sent from Team to Team are used. See Section 5 for the message formats.
- The narrative will include the details of attacks made and agents selected by both teams.
- Both sides detect the loss of the other by a read failure. Write calls could also fail, but your program should ignore these failures and wait for the associated read failure. Such write failures must not cause your program to crash.

## 3.3 Exit Statuses

All messages are to be printed to `stderr`.

| Condition | Exit | Message |
|---|---|---|
| Normal exit due to game over | 0 | |
| Wrong number of arguments | 1 | `Usage: 2310team controllerport teamfile`<br>`or: 2310team wait teamfile sinisterfile`<br>`or: 2310team challenge teamfile sinisterfile targetport` |
| Unable to open sinister file for reading | 2 | `Unable to access sinister file` |
| Contents of sinister file are invalid | 3 | `Error reading sinister file` |
| Unable to open team file for reading | 4 | `Unable to access team file` |
| Contents of team file are invalid | 5 | `Error reading team file` |
| Port number is invalid | 6 | `Invalid port number` |
| Cannot open a connection to the controller | 7 | `Unable to connect to controller` |
| Cannot open a connection to the waiting team (ONLY used during `challenge` mode) | 8 | `Unable to connect to team` |
| Controller disconnected early (NOT used during `wait`/`challenge` mode) | 9 | `Unexpected loss of controller` |
| Other team disconnected early (ONLY used during `wait`/`challenge` mode) | 10 | `Unexpected loss of team` |
| Problem encountered while parsing message, etc | 19 | `Protocol error` |
| System error (eg. malloc failure) | 20 | `System error` |

Table 2: Exit statuses for team

# 4 Controller

## 4.1 Invocation

The parameters to start the controller are:

- The height and width of the grid of zones.
- The path to the sinister file.
- Simulations to run, given as triples of: the number of rounds in the simulation, port number and number of teams expected on that port. If the port number is a dash, then the port number should be chosen by the OS. eg: `3 - 5 9 3500 8` would run two simulations, the first on an OS assigned port requiring 5 teams with three rounds and the second on port 3500 requiring 8 teams and taking nine rounds.

For example:
`2310controller 3 7 files/sinister3 3 - 4`
would create a grid of zones 3 high and 7 wide. Agent capabilities will be read from `files/sinister3`. The controller will wait for 4 teams to connect on a port chosen by the OS. The simulation will run for 3 rounds.

As soon as the controller has checked its arguments, it should print out (on a new line) the port used for each simulation (in the order given on the command line). For example:
`2310controller 4 4 files/sinister2 7 - 2 6 7502 8 6 - 5`
might print:

```
59010
7502
55033
```

Another run with the same parameters might print:

```
54323
7502
60010
```

## 4.2 Sequence

The controller will run until all simulations have completed (successfully or unsuccessfully). Each simulation will do the following:

Once the required number of teams have connected, the controller will run the required number of rounds. The simulation environment will consist of a grid of zones and a number of teams (each team being in a zone). In each round of the simulation, any teams in the same zone will battle.

The teams will then receive a `battle` message to tell them which other teams they are going to fight. The ports listed in the `battle` message should be arranged as follows:

1. Teams in the simulation should be sorted in lexicographic order by name (the first in sequence being Team 0).
2. Ports are to be listed in their team number order.
3. Each team will be told to challenge all teams with a higher team number than themselves, not including the last team. The last team will challenge everyone. For example: Suppose 4 teams must battle in a given zone (*t0*, *t1*, *t2*, *t3*), where *t0* has the lowest team number.
    - *t0* will challenge *t1*, *t2*
    - *t1* will challenge *t2*
    - *t2* will not challenge anyone
    - *t3* will challenge *t0*, *t1*, *t2*

Upon receiving a `battle` message, a team will display a message indicating their location:
`Team is in zone x y`
where `x y` is replaced with the coordinates of the zone. The team will then engage in a battle with each of the opposition teams they were sent. Such battles follow the same sequence as described earlier in Section 3.2. After a battle is complete, the team will send either a `donefighting` message to indicate the battle completed successfully, or a `disco` message to indicate an incomplete battle. At the end of each round, each team will output their narratives in order (lexicographic order by opponent team names). Note that this is done by the `2310team` process responsible for the team, not by the controller.

When the controller has recieved enough `donefighting` messages, the round is over and it will send each team either:

- a `wherenow?` to ask for their next move, or
- a `gameoverman` message to indicate the end of the simulation.

If the controller receives a `disco` message from a team instead of a `donefighting` message, the simulation will end early. The current round will finish, then the controller will send all teams a `gameoverman` message. Teams that receive this message will proceed to exit normally.
Note:

- The borders of the grid wrap. For example, this means if a team tries to move off the top of the grid, they will reappear at the bottom.
- As per team battles, both sides detect the loss of the other by a read failure. A write failure must not crash your program.

## 4.3 Exit Statuses

All messages are to be printed to `stderr`.

| Condition | Exit | Message |
|---|---|---|
| Normal exit due to game over | 0 | |
| Wrong number of arguments | 1 | `Usage: 2310controller height width sinisterfile rounds port teams [[rounds port teams] ...]` |
| Invalid grid height | 2 | `Invalid height` |
| Invalid grid width | 3 | `Invalid width` |
| Unable to open sinister file for reading | 4 | `Unable to access sinister file` |
| Contents of sinister file are invalid | 5 | `Error reading sinister file` |
| Invalid number of rounds | 6 | `Invalid number of rounds` |
| Port number is invalid | 7 | `Invalid port number` |
| Port number is in use | 8 | `Unable to listen on port` |
| Invalid number of teams | 9 | `Invalid number of teams` |
| Problem encountered while parsing message, etc | 19 | `Protocol error` |
| System error (eg. malloc failure) | 20 | `System error` |

Table 3: Exit statuses for controller

# 5 Messages

| Message | Params | From | To | Meaning |
|---|---|---|---|---|
| sinister | This is a multi-line message | Controller | Team | Used to send the sinister file to a team |
| iwannaplay $x$ $y$ $N$ $P$ | $x$ $y$ are grid coordinates where the team wants to start. $N$ is the team name. $P$ is the port the team is waiting on. | Team | Controller | A new team wants to join the simluation |
| battle $x$ $y$ $P$ | $x$ and $y$ are grid coordinates where the team is currently located. $P$ is a space separated list of port numbers (it could be empty). | Controller | Team | Starts a new round. Connect to all those ports and challenge the team at each one. |
| fightmeirl $n$ | $n$ is a team name | Team | Team | Used by challenger to send name to team they are challenging |
| haveatyou $n$ | $n$ is a team name | Team | Team | Response to `fightmeirl` to send name back |
| iselectyou $a$ | $a$ is an agent name | Team | Team | Tell the opposition team which agent they are using |
| attack $a$ $m$ | $a$ is an agent name, $m$ is an attack name | Team | Team | Indicate an attack |
| donefighting | | Team | Controller | Sent by both teams once their battle has successfully completed |
| disco | | Team | Controller | Sent by a team when the battle finished early due to the opposition disconnecting or a protocol error |
| gameoverman | | Controller | Team | Sent to all teams to indicate the simulation is over |
| wherenow? | | Controller | Team | After a round is over, ask team which direction to move next |
| travel $d$ | $d$ is a single character direction (`N`, `E`, `S`, `W`) | Team | Controller | Direction to move after all fights (in response to `wherenow?`) |

# 6 File Formats

There are two types of file that are used by the team and controller when running a simulation or battle.

## 6.1 Team file

The members of each team, the attacks to be used in battle and the directions to move to new zones are recorded in a `teamfile`.

Here is an example team file:

```
teamdoomed
cobra slither poison
gull flap swoop swoop
flying_fox flap poison
cobra poison poison slither slither
3 4
N N E E W N W E
```

The first line gives the name of the team. The next four lines give team members and a sequence of attacks they will try to use (in order). The next line gives the coordinates which this team will start at when joining a full simulation being run by a `2310controller` (the controller will mod these by the relevant dimension). The final line gives the sequence of moves that this team will make after each round of the simulation.

## 6.2 Sinister file

The capabilities of the agents are described in a common `sinister` file.

Here is an example sinister file:

```
# Type names
bird
mammal
reptile
australian
.
# Type effectiveness strings
mammal it_worked_well ok something_went_wrong
bird it_worked_great ok not_great
australian bonza ok it_was_pretty_average
reptile it_was_effective it_was_ok it_was_ineffective
.
# Type relations
australian +bird +mammal +reptile -australian
bird -reptile +mammal
reptile -mammal +bird
mammal +reptile -bird
.
# Attacks
add_beetroot australian
add_vegemite australian
eat_egg mammal
pounce mammal
slither reptile
poison reptile
lose_tail reptile
flap bird
peck bird
swoop bird
scratch mammal
.
# Agents
mongoose mammal pounce scratch swoop
cobra reptile poison slither swoop
```

```
gull bird peck swoop flap
flying_fox mammal flap poison add_beetroot
croc australian pounce add_beetroot add_vegemite
.
```

The file consists of five sections, each ending with a `.` on a newline. The items on each line should be separated by a space. A line beginning with a `#` is a comment and should be discarded. The sections will be as follows:

Section 1 List of available types on separate lines. Each agent has a type and each attack has a type.

Section 2 Attacks will have have one of three levels of effectiveness: High, Normal and Low. Each line of this section will give a type followed by an effectiveness string for each level of effectiveness in order from High to Low. These are phrases to be used when an attack of that type and effectiveness is performed.

Section 3 Describes the effectiveness of each type against the other types.

Each line gives the type followed by a list of other types preceded by `+`, `=` or `-` indicating whether the first type has High, Normal or Low effectiveness against that type. If a type relationship is not listed, then it defaults to Normal effectiveness.

Section 4 List of available attacks. Eack line gives an attack and which type the attack is.

Section 5 List of agents. Each line gives an agent, its type and then three attacks which that agent can perform.

Since the file uses the space character as a delimiter, types, attacks and effectiveness strings can use an underscore (‘`_`’) to have it replaced by a space.

### 6.3 Banned functions/features

One of the aims of this assignment is to operate with threads and TCP networking. Your programs are not permitted to start additional processes or create additional files on the filesystem or make use of IPC pipes. Further, you are not permitted to make use of gnu language extensions or any of the following:
`__attribute__, getdelim, getline, setjump, longjump, goto, select`.
You are not permitted to use posix regex features.

You are also not to use non-blocking IO nor `setbuff()` or equivalents to switch off buffering on `FILE*`.