MechMania 20 Home

Mechmania XX

Metamania

It is the year 3018, and Mechmania is celebrating its 1024th year in operation! To celebrate, the androids running Mechmania are recalling a simpler time, in which programmers were actually made of flesh and bone, which severely hindered their ability to program. They've been watching a lot of old NSA video footage taken during past Mechmanias, and would like to try and understand what it was like to be one of those strange creatures that lived and 'died' in a time before the singularity. To do so, they've wired up a series of 'meat-bag androids', the DNA of which was recovered from saliva off of a burrito buried in the ground near the ruins of what was once the Siebel Center. Having gone without use for decades, the bioengineering equipment has left the meat-bags without the ability to think for themselves. Hence, the Reflections Projections 1024 staff has decided that they will write their own intelligence for the meat-bags, and have conscripted you to do so! Whichever android team can coerce the meat-bags into performing the best will win the grand prize of .0001 bitcoin, currently valued at $3,735,928,559.

Piazza

About the game

Quick start guide

For the most up to date documentation go to

https://www-s.acm.uiuc.edu/confluence/display/mm20/MechMania+20+Home

Actions

Actions and how to use them.

Each turn your program will control your team by sending them a list of actions to be performed. (See how this is done here)

The server will return a list of the actions you attempted in the "responses" array in addition to "success" and "reason" fields for each action.

Before taking an action, each person will try to sit down in any available seats. Seats give bonuses to most actions.

The priority of an action is determined first by its speed, so actions with higher speed will occur before actions of lower speed. If two people attempt an action with the same speed, then the one who has the least hunger and fatigue will act first.

Action (Speed)

- Move (80)
- Eat (80)
- Distract (70)
- Sleep (60)
- Code (60)
- Theorize (50)
- View (50)
- Wake Up (40)
- Spy (20)

Code

Speed

60

Action Description

Tells a Person to work on programming the team's AI. Coding for multiple turns is more effective, up to a limit. Sitting down while coding is twice as effective. (You will sit down automatically if/when any chairs are available in the current room.) Certain room configurations can also increase its effectiveness.

JSON Example

```
{
"action":"code",
"type": "implement",
"person_id": 0
}
```

JSON Parameters

person_id:

The Person performing this action.

type:

The type of coding to be done. Valid types are: "refactor", "test", "implement", and "optimize". Their effects can be seen in the table below:

	refactor	test	implement	optimize
optimization			-	+
stability		+	-	
complexity	-		+	+
implementation			+	

Detailed formulas

detrimentfromfatigueandhunger:

This is a modifier that decreases as you get too hungry or too tired. It is .5 * (1/fatigueabove60) + .5 * (1/hungerabove60), meaning that there is no detriment until your fatigue or hunger go over 60, and at that point it starts to decrease that value linearly down to 0 at 100 hunger or fatigue.

effectiveness:

This is a modifier that affects all coding actions. It is the (room bonus) * (.5 if they are standing) * (turns coding bonus that increases by .5 each turn for 20 turns to a max of 10) * (detrimentfromfatigueandhunger)

refactor:

complexity -= member's refactor value (see archetype) * their coding effectiveness

test:

stability += ((member's test value (see archetype) * their coding effectiveness) / (complexity / 10)) / 100

implement:

implementation += (member's codingProwess value (see archetype) * their coding effectiveness) / (complexity / 10)

complexity += (member's codingProwess value (see archetype) * their coding effectiveness) / (complexity / 10)

optimization -= ((member's codingProwess value (see archetype) * their coding effectiveness) / (complexity / 10)) / 10

stability -= ((member's codingProwess value (see archetype) * their coding effectiveness) / (complexity / 10)) / 200

optimize:

complexity and optimization += by (member's optimize value (see archetype) * their coding effectiveness) / (complexity / 10))

Distract

Speed

70

Action Description

Tells a Person to distract another Person. Will fail if they have already moved, otherwise it causes their action to fail.

If someone has been distracted, their actions will fail and their acted field will equal 'distracted'.

JSON Example

```
{
"action":"distract",
"victim": 3,
"person_id": 0
}
```

JSON Parameters

victim:

The Person to be distracted

person_id:

The Person performing the distraction.

Eat

Speed

80

Action Description

Tells a Person to eat if there is a snack table in the room they're in.

This action decreases a Person's Hunger Level by 10x the rate at which a person gains hunger.

JSON Example

```
{
"action":"eat",
"person_id": 0
}
```

JSON Parameters

person_id:

The Person performing this action.

Move

Speed

80

Action Description

Tells a Person to move from one room to another.

When entering a room, you start off standing.

JSON Example

```
{
"action":"move",
"room": "5",
"person_id": 0
}
```

room:

The Room to move into. It must be adjacent to the room the person is currently in.

person_id:

The Person being told to move.

Sleep

Sleep

60

Action Description

Tells a Person to go to sleep. On success, this action reduces a person's Fatigue.

Notes

The Fatigue reduction per turn is based on the number of game ticks per hour.

It takes 6 hours to go from full to empty Fatigue in an empty room while sitting in a chair. Every other Person in the room adds 30 minutes to this number (regardless of their sleep status), up to a maximum of 12 hours total.

You must be sitting to voluntarily sleep. If you reach 100% Fatigue, you will uncontrollably fall asleep (in your chair if you're in one, otherwise on the floor) at the end of your turn. Sleeping on the floor is half as effective as sleeping in a chair, but will also take a maximum of 12 hours total.

If a Professor is in the room, all forms of sleeping are twice as effective as they would be otherwise. (The worst-case sleeping time while sitting is still 12 hours.)

When people are supposed to wake up, they wait until the end of your turn to do so. In order to wake up, either another Person must wake you or you must sleep until you reach 100% hunger. **Note that you do not automatically wake up at 0% fatigue.**

JSON Example

```
{
"action":"sleep",
"person_id": 0
}
```

JSON Parameters

person_id:

The Person being told to sleep.

Spy

Speed

20

Action Description

Tells a Person to spy on everybody in their current room. This would do nothing in a room with only team mates.

Theory += spy bonus * (enemies in the same room coding + 2 * enemies in the same room theorizing)

Note that this does not count teammates.

JSON Example

```
{
"action":"spy",
"person_id": 0
}
```

JSON Parameters

person_id:

Defines the Person performing this action

Theorize

Speed

50

Action Description

Tells a Person to think about how the AI should work. Sitting down makes this twice as effective. (You will sit down automatically if/when any chairs are available in the current room.) Certain room configurations can also increase its effectiveness.

The exact formula is as follows: It increases theory by that team member's theorize value (see archetypes) * (.5 if standing) * (room bonus) * (detrimentfromfatigueandhunger)

JSON Example

```
{
"action":"theorize",
"person_id": 0
}
```

JSON Parameters

person_id:

The Person who should theorize.

View

Speed

50

Action Description

View the practice games to determine other teams' scores. Only works in a room with a projector after the practice games have begun. (They begin when the game is half over.)

JSON Example

```
{
"action":"view",
"person_id": 0
}
```

JSON Parameters

person_id:

The Person performing this action.

Return JSON Example

```
{
"action": "view",
"person_id": 0,
"message": { 0 : 12.4829, ... },
"success": true
}
```

Wake Up

Speed

40

Action Description

Tells a Person to wake another sleeping Person up.

JSON Example

```
{
"action":"wake",
"victim": 5,
"person_id": 0
}
```

JSON Parameters

victim:

The Person to be woken up

person_id:

The Person doing the awakening

Communicating with the Server

Connecting to the Server

To connect to the server, set up a socket connection using TCP. Unless a different hostname and port was specified as a command-line argument, connect to port localhost 8080.

All messages sent to and from the server are formatted as a JSON ASCII string and delimited by a newline.

Initial Connection

The first message sent after connecting to the server should include team name, and the names and archetype of all team members. There is a team size limit of 3 members.

```
{ "team":"test", "members":[ {"name":"test", "archetype":"Coder"}, ... ]}
```

Server Response

The response to the initial connection has two fields, "status" (either "Success" or "Failure"), and "errors" (a list of errors that occurred in case of failure)

```
{
"status": "Success",
"errors" : [],
"team": {name: {<person>}, ...},
   "team_name": "team"
"turns_per_hour": 60
}
```

Notes:

- *"errors" is only included on failure and "team" is only included on success.*
- *If a player fails to connect they may try again until the time limit ends, at which point they will be disqualified.*
- *"turns_per_hour" is the number of in game turns per in game hour. There will always be 24 in game hours per game.*

Sent turn data

The turn data sent should be an array of actions:

```
[ {<action>}, {<action>}, ... ]
```

If a person does not get assigned an action, they will do nothing.

Received turn info

You will receive a dictionary containing errors (such as invalid moves), warnings (such as valid moves that could not be executed), the progress of your AI, and map info:

```
{
"aiStats":
{
"complexity": 1.0,
"implementation": 0.0,
"optimization": 0.0,
"stability": 1.0,
"theory": 0.0
},
"errors": [],
"people": {person_id: {<person>}, ...}
"map": {room_id: {<room>},...},
"messages":
[
{
"action": "dummy",
"person_id": "0",
"message": "Human readable",
"reason": "Dummy not a valid action",
"success": false
},
{<action>}
...
]
"events": []
}
```

aiStats

Current stats for your team's AI.

errors

Information about attempted actions that failed to execute, if any.

people

Current stats for all the team members in the current game. You get complete information about people that are on your team, partial info on people in the same room, and basic info on everyone else.

map

Dictionary of rooms where the key is the room ID.

messages

List of responses from actions. On failure, will only contain a "reason" field.

events

Events are announced to all teams. The event list contains dictionary objects with two keys: "name" and "message". The following table shows what the values for name and message field would contain for valid events.

Name	Message
"PROFESSOR"	The id of the room that the professor is in.
"NOPROFESSOR"	The id of the room that the professor left.
"PRACTICE"	"The projectors are now showing practice games"

Notes: These are only sent out once! Take note of them.

Game Over

The response contains exactly one field indicating whether the client won the game or not. Upon receiving this message, the client should shut down.

```
{ "winner": false }
```

Game info

Mechmania XX: Metamania!

Background

You are in charge of a team coding for Mechmania! This is a turn-based game, played on a map that is comprised of rooms, which you can think of as nodes on a graph. You start by telling the server what members you have (or want) on your team, and then each turn you assign actions to those team members to execute. Each player has one action per turn that they can execute, assuming that they are not asleep or otherwise incapable of performing that action. For the most part, your team members are going to code their AI over the course of 24 (in-game) hours to code their AI. When those hours are up, the team with the highest scoring AI wins.

Win Condition

The formula used to score the AI is **stability * ((optimization) * C + (implementation - optimization) * c)** where C is a large constant and c is a small constant.

Game Runner

Overview

./gamerunner.py will run the game. It only works when your run.sh file is set up correctly.

Optional arguments

Command	Shortcut	Description	Default
--help	-h	Show a help message and exit.	
--port PORT	-u PORT	The port which the server uses.	8080
--debug-view	-w	Show debug information.	
-m MAP	-m MAP	The map file the game parses from.	rooms_full.bmp
--mapOverlay OVERLAY	-o OVERLAY	The map overlay file the visualizer uses.	rooms_overlay.png
--log LOG	-l LOG	The file the game log will be written to.	serverlog.json
--teams TEAMS	-t TEAMS	The number of teams.	2
--client CLIENT	-c CLIENT	The client to be run. Example: ./gamerunner.py -t 3 -c myClient -c test_clients/python The gamerunner will fill any unspecified slots with the default test client.	
--defaultClient DEFAULTCLIENT	-d DEFAULTCLIENT	The client to be run if no specific clients are given.	../test-clients/python
-verbose	-v	Print out one player's standard output.	
-veryVerbose	-vv	Print out every player's standard output.	
--show	-s	Show the visualizer in a window that displays the game state.	
--cached-map	-C	Use a precomputed cached map (filename: map.cache) if one exists.	

Getting Help

Piazza

If you have questions or are stuck on something, you can access the Mechmania 20 Piazza. (access code: "acm1404", without quotes).

Questions posted here are questions that everyone in the competition can see and search through.

One-on-one help

If you need one-on-one help, post on Piazza or ask a lab assistant.

Quick Start Guide

Overview

This page is a central list of resources that are helpful while coding your AI.

Step-by-step Quick Start guide

A easy list of things you should do to get started.

1. Log into your bit bucket account and setup your SSH key.
2. Pull down the Mechmania Repo.
    a. You should find a folder of test clients and a folder for the game.
3. Make a run.sh in the root of your folder (based off of the similar run.sh for the test client).
    a. Make it executable with "chmod +x"
4. Read the communicating with server and rules pages.
5. Start coding your AI!

Game Info

For info on the game, go to the about page.

Playing The Game

Here is a guide explaining how the client and server communicate.

Here are instructions on how to launch the game.

Clients

We have provided several stubbed out clients that you can use when writing your AI. Alternatively, you may also create your own client. If you choose to go this route, your client must be able to run on your provided lab machine. **If your client does not run during the final competition, you will be disqualified.**

Provided clients

The clients folder includes all of our provided clients, including the run.sh used to run them.

This year, the following provided clients are available to you:

- Python
- Java
- C

Making your own

Here are some pages that might be useful if you are trying to make your own client.

Once again, a client that does not run during the final competition will result in disqualification.

The Communication Guide (linked above)

How to format the run.sh file

C Client

Dependencies

You need libjansson-dev to use the c client. It is used to parse json.

What you get

main.c

Holds the main structure of the program. The main receive server information/ send turn loop is here.

ai_objects.c

Holds struct versions of all of the server json obejcts. Some of the variable have changed names and some of the objects have become simple buffers, so be sure to look at how the structs are set up here before using them. This file also contains functions for freeing all of these structs. Buffers should be initialized with malloc() (as oppose to new if you are using c++) so they can be properly free'd by free(). Any unused buffers should be explicitly set to NULL so free doesn't attempt to free some garage address.

parse_json.c

This is all the code for converting between our structs and libjansson json objects. Ideally you will not have to look at this code.

client_ai.c

This contains the extremely simple example AI we're giving you. get_initial_message sets up the initial team. get_first_turn returns your actions for the first turn. get_turn returns your actions for every other turn. Currently get_first_turn does nothing and get_turn moves each of your team members to a random adjacent room. The reason get_first_turn and get_turn are separated is each are supplied different information in a different format (because of the way the server protocol was defined).

utils.c

This contains a few utility functions for networking, string, and file manipulation.

Java Client

The java test client can be built using the ant script in the accompanying folder. First run mkdir -p bin/edu/acm/uiuc/mm20/objects. Then type 'ant' and it will export the client as a jar with the correct filename (run.jar) that run.sh runs.

If you modify this ant file due to adding dependencies, for example, then make sure that the paths are relative to the folder it is in. The build.xml file isn't very large, you can do this by hand.

NOTE: ant apparently requires a bin folder to exist, so run it first in an ide like eclipse to generate bin and ant will work fine. We are trying to develop a workaround.

For the moment, if you're using eclipse, you can open the project by using new-> project -> java project. At this screen, uncheck using default folder and use test-clients/java instead.

Python Client

Running Instructions

The Python Client can be run by using ./run.sh in the test-clients/python folder.

What you get

The base Python client tells the team members to theorize, code, or test based on their archetype, and to eat if they get too hungry. Change the behavior of this client by editing the setActions function.

Variables

members - A dictionary of person objects of the format:

```
{person_id : {<person>}, ... }
```

value - The JSON server response for the turn after it has been turned into an object. You'l want to pull values out of this to inform your decisions.

Functions

updateMembers - Taking the value returned by the server that turn, we iterate through the people dict and update the stats of anyone on our team. This means that we have a dictionary where the key is their idea and the value is another dictionary corresponding to that person's values and seen in the json example on the person page.

setActions - Taking the list of members on our team and the value returned by the server, return a list of actions for each member. We simply iterate over every member and set their actions appropriately.

Rules

The Mechmania rules for this year are as follows:

COMPETITION LOCATION

To ensure fairness for each participant and team competing in MechMania XX, MechMania Staff asks that you adhere to the following guidelines during the competition. Failure to adhere to these guidelines will result in the disqualification of your entire team from the competition. MechMania staff always have discretion.

- Please be respectful to other participants/teams. Keep the competition friendly!
- Each person can be a member of one and only one team.
- If you encounter a bug in the code provided to you, you must report it to the MechMania Staff immediately. Exploiting a bug in the MechMania server is cheating.
- You may not use a library that was designed or created specifically for MechMania.
- External public opensource libraries that were not written for MechMania are allowed.
- Your team must write the majority of code that runs your AI Client.
- Please do not bring food into the MechMania lab, and keep the lab area clean.
- Your AI Client must be committed to the provided repository before the competition closes. Code committed after the deadline will not be run.

NETWORK ACCESS

Competitors that are not students/staff at the University of Illinois should use the “IllinoisNet” wireless SSID, with the credentials located on the back of their name badge. The “UIPublicWifi” SSID may block ports that are necessary or useful for MechMania from crossing the firewall; the firewall on the IllinoisNet network is less strict than all other available networks.

COMPUTER LAB SPACE

The computer lab equipment that is available for use during the MechMania competition is property of the University of Illinois and should be treated with respect. Please clean up your workspace after use, and keep food outside of the lab.
If you are using your own laptop, you may log into a MechMania computer with ssh <username>@mechmania.acm.illinois.edu
Please note that all storage is local to the MechMania computers (including remote workstations) - storage is not synchronized between workstations. If you wish to share files or code, it is recommended you use your team’s Git repository to do this.

FOOD AND MEALS

Meals are provided as part of the MechMania registration fee. Your MechMania name badge will allow you to join meals. Snacks and meals will also be brought down to the MechMania competition area. Please keep the lab space clean, and

refrain from bringing food into the lab.

run.sh

Overview

Competitors must provide a run.sh file which starts their client in a game-ready state. Examples can be found in the provided test clients.

Instructions

Your client should run when './run.sh' is executed in the terminal when in the client's folder. It should take two arguments hostname and port. **IF IT DOES NOT, WE WILL BE UNABLE TO RUN YOUR CLIENT AND YOU WILL BE DISQUALIFIED.**

Example

```
#!/bin/bash

if hash python2 2>/dev/null; then

        python2 test_client.py $1 $2

    else

        python test_client.py $1 $2

    fi
```

Problems?

If you will get a "bash: ./run.sh: Permission denied" error, run the command "chmod +x run.sh" to give it execution permissions. If this does not work, contact a lab assistant.

Running on your own machine

Overview

This page describes the steps required to get Mechmania 20 running on your own machine.

Dependencies

These are the libraries required to run the game.

- pillow
- pygame

Install them using one of the following commands:

Linux package installer

- sudo apt-get install **library**

Mac package installer

- brew install **library**

Python package installers

- pip install **library**
- easy_install **library**

Stuck?

Post on Piazza. If you need individual assistance, let us know and we'll come take a look.

Types and Objects

AI

The AI is represented by various statistics that change based on the actions that team members take. These are the stats as well as their starting values:

```
{
    "optimization": 0.0,
    "stability": 1.0,
    "complexity": 1.0,
    "theory": 0.0,
    "implementation": 0.0
}
```

Note: Unless otherwise stated, stats have a minimum value of 0 and no maximum value.

optimization:

This improves the AI's performance, but is capped by implementation.

stability:

A value from 0 to 1 inclusive that indicates how stable and bug-free the AI is. The higher, the better.

complexity:

This is how complex and confusing the code has become. As this increases, progress becomes slower. The minimum value for this stat is .25 * the implementation value.

theory:

This is how much of the AI you've thought through. It caps implementation.

implementation:

This is how much of the AI has been completed. It is capped by theory.

Here is a table showing how different actions affect the AI:

	refactor	test	implement	optimize	spy/theorize
optimization			-	+	
stability		+	-		
complexity	-		+	+	
theory					+
implementation			+		

Projector

Overview

A Projector is a piece of furniture that allows anyone in its Room to view practice games.

Functionality

Practice games begin halfway into the competition (announced in the events field) and allow people to gain information on their competitors using the view action.

Room

Object description

This object represents a particular room in the game. Rooms can have various things (such as chairs and snacktables) that Persons can interact with.

Rooms can also give passive bonuses. For every teammate in the same room, a person's coding and theorizing effectiveness increases by 10%. For every professor in the same room, a person's coding and theorizing effectiveness and sleep rate increase by 100%.

JSON Example

```
{

"room": "2"

"connectedRooms": [
"11",
"5"
],
"peopleInRoom": [0, 7, ...]

"resources": ["FOOD", ...]

"seatsTotal": 3
"standsTotal": 3

"seatsAvailable": 1

  "standsAvailable": 3

}
```

JSON Attributes

room:

The unique room name. Currently this is a two-digit room number.

connectedRooms:

A list of the names of the adjacent rooms. A person in this room can walk to any adjacent room in a single action.

resources:

A list of available resources. Possible resources include "FOOD" if the room contains a snack table, "PROFESSOR" if there is a professor, and "PROJECTOR" if there is a projector.

peopleInRoom:

A list of the people ids in this room.

Snack Table

Overview

Snack Tables are a piece of furniture and the sole source of food in the game. A room has a snack table if they have "FOOD" in their resource list.

Team Member (Person)

Object Description

A Team Member (sometimes referred to as a Person) represents an individual member of a human player's team.

To create a team member, supply the team member's name and archetype on game startup. A team member's archetype determines their stats. The list of archetypes and their stats can be found at the bottom of the page.

Each real-life team can have a maximum of 3 in-game team members.

JSON Example:


```
{
"acted": null,
"asleep": false,
"fatigue": 50.0,
"hunger": 40.0,
"sitting": true,
"location": "12",
"name": "test2",
  "team": 1
"person_id": 4,
"archetype": "Coder",
"stats":
{
"codingProwess": 10,
"optimize": 5,
"refactor": 7,
"spy": 5,
"test": 7,
"theorize": 3
}
}
```


Note: Players not on your team do not necessarily give you all their information.

JSON Attributes

Person_id:

A unique ID given to you by the server. Use this to specify the person when sending actions.

Name:

The name you gave your person. This isn't used by the game engine, but it will be shown on the visualizer.

Inappropriate names will be sanitized.

Acted:

The action that this person just performed. If the person was distracted, then their acted field will equal "distracted" instead of the action they attempted to do.

Asleep:

Is your team member asleep? (Boolean).

Note: once a team member falls asleep, they will only wake up if they are woken up by another person or their hunger hits 100%. ***In particular, they will not wake up just because their Fatigue reaches 0%.***

Fatigue:

The sleepiness of this person. When a team member's Fatigue hits 100, they fall asleep. Takes 16 in game hours to go from 0 to 100. Starts at 50. This CANNOT go negative!

Note: If team members have both maximum Fatigue and maximum Hunger, they alternate between waking up and falling asleep. This allows you to do one action every other turn.

Hunger:

The hunger level of this person. When a team member's Hunger hits 100, they can only move and eat. Takes 8 in game hours to go from 0 to 100. Starts at 0.

Note: If team members have both maximum Fatigue and maximum Hunger, they alternate between waking up and falling asleep. This allows you to do one action every other turn.

Location:

The name of the room this person is in.

Team:

The team number the player belongs to.

Sitting:

Whether the team member is sitting down or not.

Archetype:

The archetype that the team member falls into. See table at the bottom of the page.

Stats:

The stats applicable to that archetype. See table at the bottom of the page.

Data Visibility Levels

Every person:

person_id, *team*, and *name* are visible.

Same Room:

person_id, *team*, *name*, *asleep*, *acted*, *sitting*, and *location* are visible.

Same Team:

All attributes are visible.

Archetypes information:

```
"archetypes": {
"Coder": {
"codingProwess": 10,
"refactor":7,
"test":7,
"optimize":5,
"theorize":3,
"spy":5
},
"Theorist": {
"codingProwess": 5,
"refactor":5,
"test":7,
"optimize":5,
"theorize":10,
"spy":5
},
"Architect": {
"codingProwess": 7,
"refactor":10,
"test":10,
"optimize":5,
"theorize":5,
"spy":5
},
"Informant": {
"codingProwess": 3,
"refactor":5,
"test":5,
"optimize":5,
"theorize":7,
"spy":10
}
```