Wumpus World
AI Project

By
Abdullah Younis

For
CS-171
Introduction to Artificial Intelligence

I. Table of Contents

II. Executive Summary

This report details the Wumpus World game, coding shells, tournament program, and recommended classroom implementation. This Wumpus World Project is designed for the University of California Irvine's CS-171 class. Most importantly, this project is meant to be educational; a student who diligently works on this project will learn about knowledge-based agents in a hands-on environment.

The Wumpus World game consists of a dark cave and an agent, which makes its way to a heap of gold avoiding bottomless pits and the Wumpus, a dangerous monster. The coding shells implement the game while making room for students to code their own rational agent. A student has the option of choosing to code in C++, Java, or Python. Once the student selects his or her language, he or she will begin on the assignment, which is to write an intelligent agent that will solve the Wumpus World problem while trying to maximize its performance measure.

Every student is supplied with ample resources to simplify compiling, testing, and finalizing the project. This is done to ensure most of a student's time is spent on implementing learning material rather than auxiliary processes. To achieve this, a script for compiling and packaging the project for submission is included in each shell. A world generator will be supplied as well to quickly generate large sets of test worlds.

After the submission deadline, all of the student-written agents will compete against each other in a tournament. The tournament program will generate a set of test worlds, and all of the student-written agents will be tested on these worlds. Each agent's average score is recorded, and a scoreboard is produced by the tournament program. The idea of competition is meant to excite students to go above and beyond the requirements for full marks.

There will be many ways to implement this project into a classroom setting, however, a series of recommendations is made in this report. Some deviations from the recommendations will require some tweaking in the tournament program. These hard-coded policies will be noted alongside a detailed description of how to change the code to the desired functionality.

A project booklet is made available to the students, which details the assignment, the game, the shells, how to get started, how to connect to Openlab, and the tournament process. Some deviations from the recommendations will require an update to some portion of the

student booklet. These types of policies will be noted. Additionally, instructors will have a small booklet detailing how to operate the tournament and interpret its output.

III. Project Qualities

Parallel

This project aims to provide parallel shells for the students. A student, who is fluent in Java, C++, and Python, will find no best shell to choose. A student, who is fluent in Java, C++, and Python, will find no advantage to choosing one shell versus another. All of the shells are built using the same design, all of the shells interface with the student the same, and all of the shells interface with the command line the same. The tournament treats all shells equally.

Robust

The tournament program and shells aim to be robust. Any erroneous inputs will produce a helpful response. Any erroneous input will be handled by a smooth exception flow. A student will never be without a clue as to why his or her input didn't work.

Accurate

The tournament program and shells are accurate when calculating score, average score, and standard deviation. This requirement is essential because grading will be based on the performance measure. Incorrect calculations will lead to incorrect and potentially harmful grades. A student’s effort throughout this project should be rewarded appropriately and having accurate results is a necessary condition.

Reproducible

The tournament program needs to be reproducible since grades are based on the output of the tournament. If a student has a dispute with his or her score, the tournament program should be able to reproduce the exact tournament and scoring process. This is accomplished by the tournament including the set of generated worlds in its output, as well as, any messages from the compiler, the executor, or other parts of the tournament. These worlds should be published alongside the scoreboard; this way, a student may run their program on the tournament worlds and expect the same score on the scoreboard. Since ample log files are created, any discrepancies can be handled appropriately.

Installable

The coding shells are designed to be user-friendly when installing and getting started. Once downloaded, a student picks their language and starts coding. There is a makefile script included with each shell, which makes compiling the project one command. When finished, another command can be used to package the project for submission.

Secure

There is a threat model detailed further in this report. The tournament program and shells will take appropriate measures to handle any threats. Confidential student information is handled by the tournament, so security of such information is important. Additionally, cheating must be handled appropriately if discovered by the tournament

Maintainable, Operable, and Survivable

This project aims to be survivable for years to come. For this to be the case, this project needs to be easily modifiable and maintainable. Any instructor or teaching staff implementing this project will find the tournament to be easily operable and ample documentation if he or she is to get stuck.

IV. Game Mechanics

The Wumpus World is a cave consisting of rooms connected by passageways. Lurking somewhere in the cave is the terrible wumpus, a beast that eats anyone who enters its room. The wumpus can be shot by an agent, but the agent has only one arrow. Some rooms contain bottomless pits that will trap anyone who wanders into these rooms (except for the wumpus, which is too big to fall in). The only mitigating feature of this bleak environment is the possibility of finding a heap of gold. A concrete definition of the game is given by this PEAS description:

Performance Measure

The performance measure of an agent is an integer score calculated based on the following:

- Start at 0 points
- -1 point for each action taken
- -10 for using the arrow (additional to the -1 point)
- -1000 points for falling into a pit or being eaten by the wumpus
- +1000 for climbing out of the cave with the gold

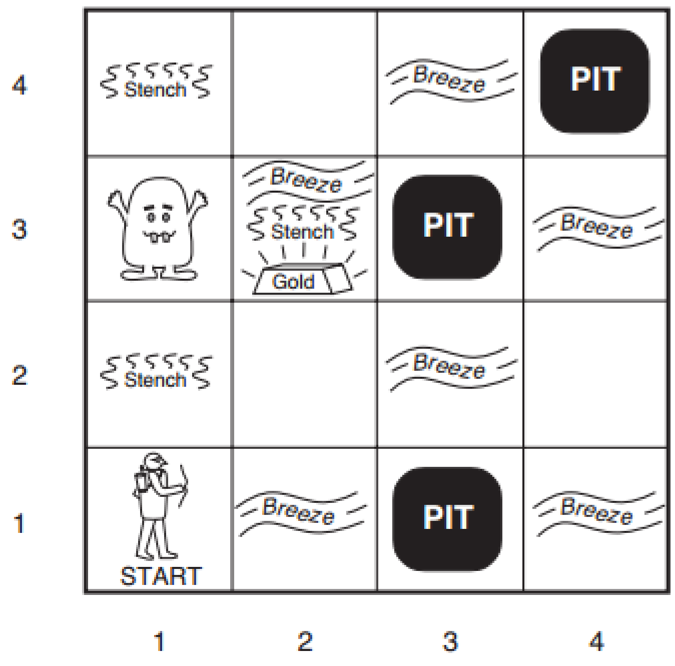
The game ends either when the agent dies, when the agent climbs out of the cave, or when the agent's score goes below -1000.

Environment

The environment can be classified as partially observable, deterministic, sequential, static, discrete, and single agent.

- An NxM grid of rooms, where $4 \leq N, M \leq 10$.
- The agent always starts in the bottom left square (1, 1), facing to the right.
- The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square.
- Each square other than the start can be a pit, with a 20% probability.
- The agent dies a miserable death if it enters a square containing a pit or a live wumpus.

An example 4x4 Wumpus World:



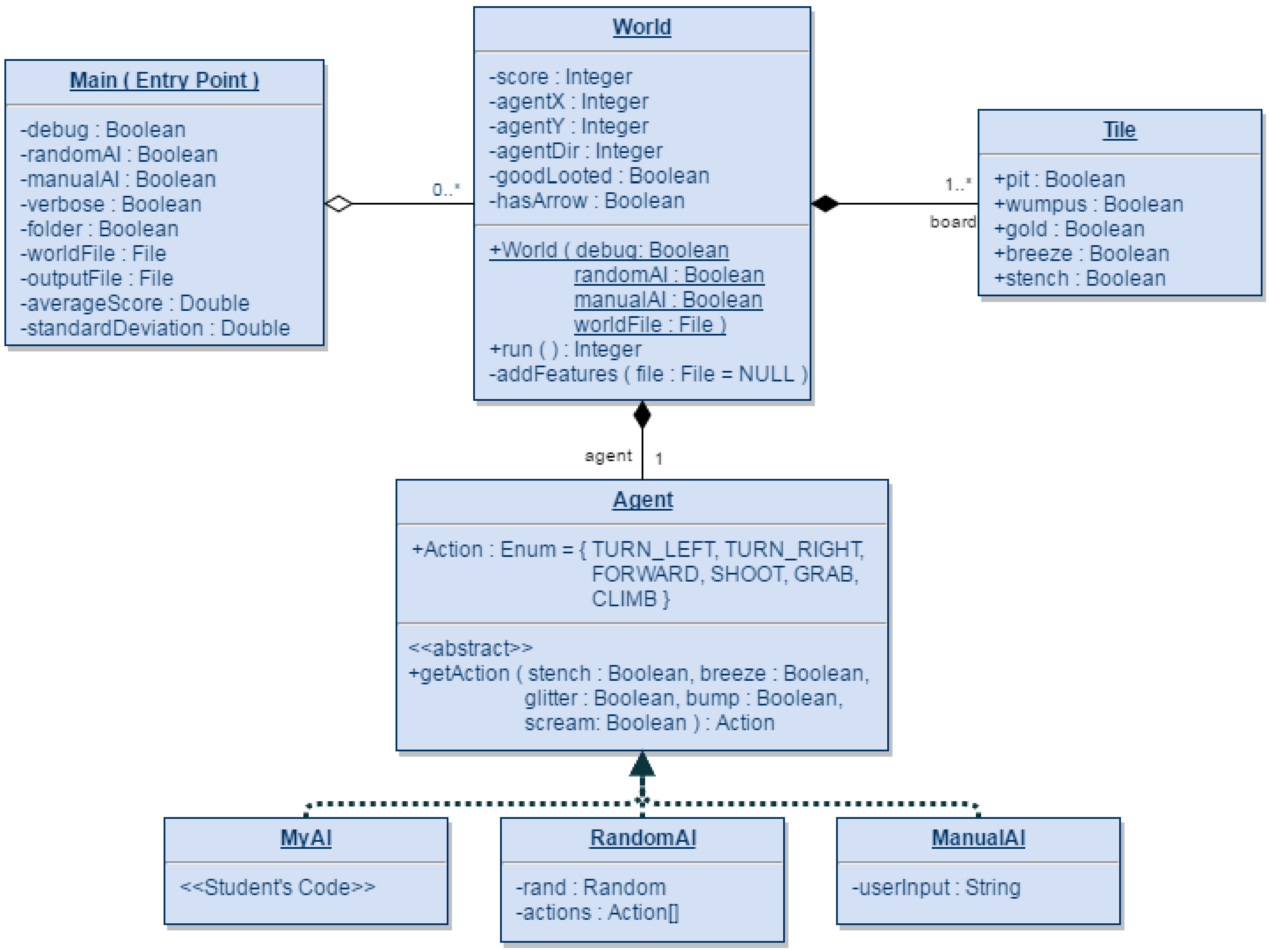
Actuators

- The agent can move **FORWARD**, **TURN_LEFT** by 90 degrees, or **TURN_RIGHT** by 90 degrees.
- The action **GRAB** can be used to pick up the gold if it is in the same square as the agent.
- The action **SHOOT** can be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits and kills the wumpus or hits a wall. The agent has only one arrow, so only the first shoot action has any effect.
- The action **CLIMB** can be used to climb out of the cave, but only from square (1, 1).

Sensors

- In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a **STENCH**.
- In the squares directly adjacent to a pit, the agent will perceive a **BREEZE**.
- In the square where the gold is, the agent will perceive a **GLITTER.**
- When an agent walks into a wall, it will perceive a **BUMP**.
- When the wumpus is killed, it emits a woeful **SCREAM** that can be perceived anywhere in the cave. This percept will only be sensed on the turn immediately after the wumpus's death.

V. Shell Design



This UML Diagram serves as the basis for all three shells. The main file acts as the driver; this portion of the shell will interface with the command line, correctly instantiate all necessary World objects, collect statistics, and print output results to the desired location. Nevertheless, the main components of the design are the World and Agent classes.

The World class represents the Wumpus World environment. It has many purposes, one of which is to act as the program engine. The run function progresses the agent through the game correctly. The world class accomplishes this by supplying the agent with accurate percepts and asking for the agent's action at every turn. This same function will terminate the game and report a score when necessary.

This class also serves as the guardian of information by privately holding important agent and board data. The board is simply a private, double array of Tile objects; a Tile object contains appropriate information for each room in the cave.

The World class can be constructed with or without a world file. In the case where no world file is specified, a random 4x4 board is constructed. If a world file is specified, the board is constructed as specified in the file. If the debug flag is set to true, detailed messages will be printed to the console at every turn. Based on constructor parameters, one of the three available agents is initialized.

The abstract Agent class is realized by MyAI, RandomAI, and ManualAI. Every Agent class is required to implement the abstract function getAction. This function returns the action decided upon by the agent and the percepts. The percepts are listed as boolean parameters to the getAction function. RandomAI is an agent that returns a random action at every turn, except that it will grab if it perceives glitter. ManualAI yields control of the agent over to the student; this allows a student to play the game as an omniscient agent. MyAI, the default choice for an agent, is the student-implemented agent.

The World File

The format for a Wumpus World file:

```
[row dimension][tab][column dimension]
[wumpus row] [tab] [wumpus column]
[gold row][tab][gold column]
[number of pits]
[pit1 row][tab][pit1 column]
[pit2 row][tab][pit2 column]
.
.
.
[pitN row][tab][pitN column]
```

World Generator

The students will have access to a world generator, which will generate a set of worlds. A makefile script with helpful commands is supplied as well, one of which will generate the same set of worlds used in the tournament. A more detailed manual is included with the generator.

VI. Shell Manual

Name

The command line name used to invoke this program will change depending on the shells:

$$\text{Wumpus\_World} = \begin{cases} \text{python Main.pyc} & \text{if using python shell} \\ \text{java } - \text{jar Wumpus\_World.jar} & \text{if using java shell} \\ \text{Wumpus\_World} & \text{if using cpp shell} \end{cases}$$

Synopsis

Wumpus_World [Options] [InputFile] [OutputFile]

Options

-m	Use the ManualAI instead of MyAI. If both –m and –r specified, ManualAI will be turned off.
-r	Use the RandomAI instead of MyAI.
-d	Debug mode, which displays the game board after every move. Redundant with –m.
-h	Displays help menu and quits program.
-v	Verbose mode, which displays name of world files as they are loaded.
-f	Treats the InputFile as a folder containing many worlds. The program will then construct a world for every valid world file found. This will trigger the program to display average score and standard deviation instead of a single score. The InputFile operand must be specified with this option.

Operands

InputFile	A path to a valid Wumpus World file, or folder with –f. This operand is optional unless used with –f or OutputFile.
OutputFIle	A path to a file where the results will be written. This is optional.

Examples

Wumpus_World	Constructs a random 4x4 world, runs the MyAI agent on the world, and prints output to console.
Wumpus_World -m	Constructs a random 4x4 world, runs the ManualAI agent on the world, and prints output to console.
Wumpus_World -d	Constructs a random 4x4 world, runs the MyAI agent on the world, and prints output to console. After every turn, the game pauses and prints the current game state to the console.
Wumpud_World -r	Constructs a random 4x4 world, runs the RandomAI agent on the world, and prints output to console.
Wumpus_World -h	Prints the help menu and terminates.

Wumpus World Project
Introduction to Artificial Intelligence

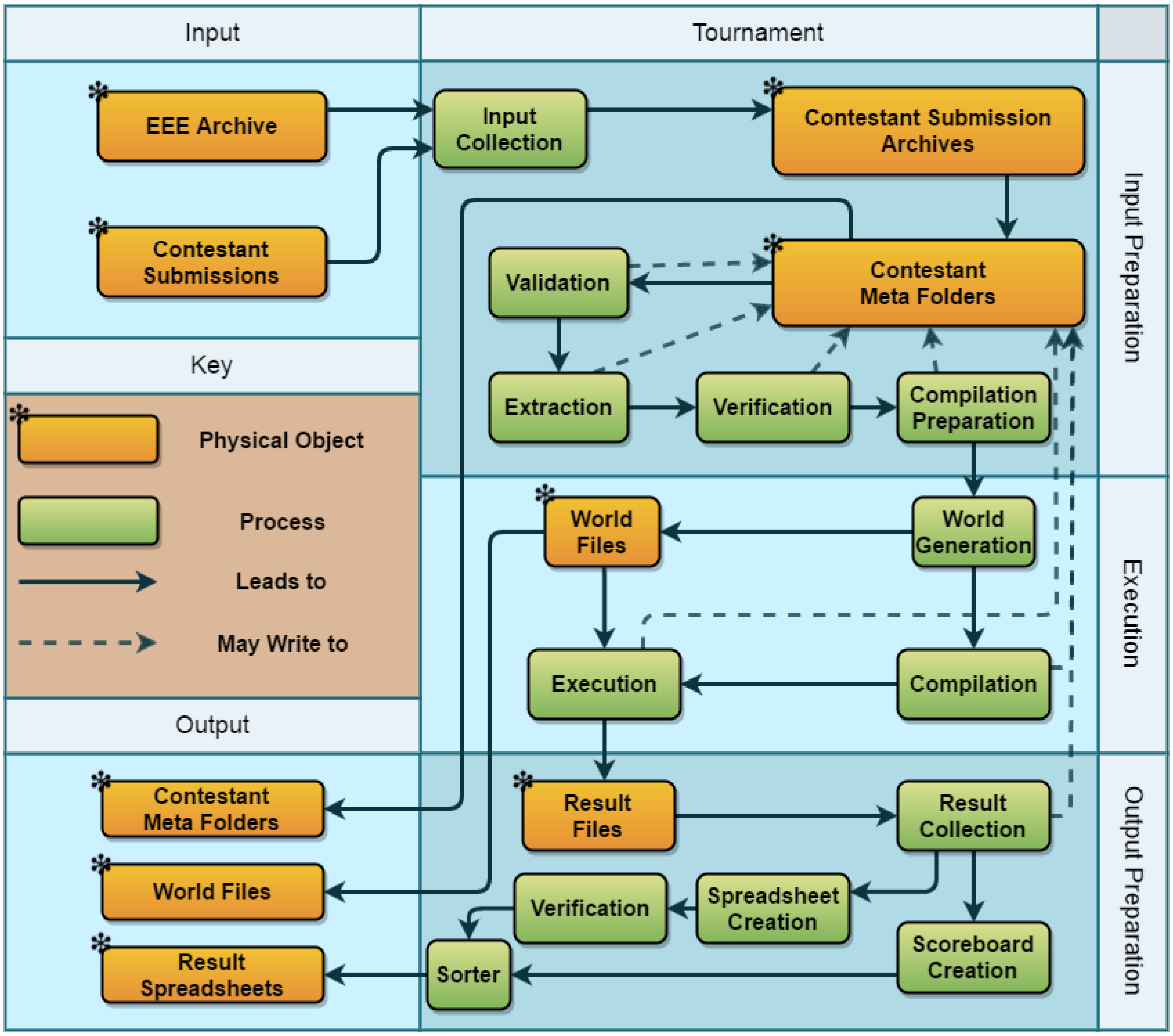
Wumpus_World -rd	Constructs a random 4x4 world, runs the RandomAI agent on the world, and prints output to console. After every turn, the game pauses and prints the current game state to the console.
Wumpus_World /path/to/world/file.txt	Constructs the world specified in the file, runs the MyAI agent on the world, and prints output to console.
Wumpus_World -f /path/to/world/files/	Constructs all the worlds specified in the folder, runs the MyAI agent on all the worlds, and prints output to console.
Wumpus_World -fv /path/to/world/files/	Constructs all the worlds specified in the folder, prints world names as they are loaded; runs the MyAI agent on all the worlds, and prints output to console.
Wumpus_World -rf /path/to/world/files/	Constructs all the worlds specified in the folder, runs the RandomAI agent on all the worlds, and prints output to console.
Wumpus_World /path/to/world/file.txt /path/to/output/file.txt	Constructs the world specified in the file, runs the MyAI agent on the world, and prints output to the output file.
Wumpus_World -rf /path/to/world/files/ /path/to/output/file.txt	Constructs all the worlds specified in the folder, runs the RandomAI agent on all the worlds, and prints output to the output file.

Notes

The Python shell uses Python version 3.5.2.

When using debug mode or ManualAI, the board will printed to the console. Each tile is represented as a full stop potentially followed by a series of characters. A ‘@’ represents the agent, a ‘P’ represents a pit, a ‘W’ represents the wumpus, a ‘G’ represents the gold, a ‘B’ represents a breeze, and an ‘S’ represents a stench.

VII. Tournament Design



The tournament program plays an important role in this project, and the above diagram represents the various steps employed throughout the whole process. Its primary job is to produce and organize scores for all contestants, which are submitted agents playing in the tournament. Nevertheless, the tournament plays a bigger role. It also verifies that all agents are following the rules of the tournament and of the project. The tournament program accomplishes all of this by breaking down the process into three parts: input preparation, execution, and output preparation.

Input Preparation

The input preparation portion of the tournament is responsible for collecting input, constructing necessary files for each contestant, and verifying that only valid contestants are allowed to play in the tournament. More specifically, input preparation consists of input collection, meta folder creation, validation, extraction, verification, and final preparations for compilation.

Input Collection

The tournament's first step involves collecting all submissions from the input folder; this step also extracts submissions from any EEE downloaded archive. An EEE downloaded archive starts with "AssignmentSubmission".

Meta Folder Creation

In this stage, all contestants receive a working directory inside a meta directory. These folders start with only the raw submission and a meta file, which serves as an important log file throughout the tournament. Each player's meta directory will be populated with more data at most every other stage of the tournament.

Validation

At this point, the submission archives undergo a validation check, which consists of filtering out invalid submissions and retrieving information from valid submission names. A contestant's submission can be invalid if it is not in "zip" format or if it doesn't contain the correct amount of underscores. Contestants with invalid submissions will be disqualified, and a report will be detailed in their meta file. Valid contestants will have their UCI Net ID, last name, ID number, and team name extracted from their submission name.

Extraction

The submission archives are extracted. Any failures here will lead to a contestant’s disqualification; the failure will be recorded in the contestant’s meta file.

Verification

Contestant's submission now undergo their final pre-execution check in the verification stage. Contestants are verified that they are following the rules of the project. Any submission must include a "pdf" report as well as valid source code of the MyAI class. If no "pdf" files are found, an error is recorded to contestant's meta file; however, if no source code is found the contestant is disqualified. In the case of a disqualification, as always, a report is made to the contestant's meta file.

Compilation Preparation

Contestant’s submission language has been determined already and MyAI classes have been extracted. The final portion of input preparation involves copying clean-slate versions of the remaining source code and placing it inside each contestant’s working folder.

Execution

The Execution portion of the tournament focuses on producing scores for each contestant. To accomplish this, it must first compile each contestant and generate a set of worlds. There is a large possibility for error in this section, so the tournament handles this portion with care.

World Generation

The first part of the execution step is to generate a set of worlds. The world generator is invoked multiple times to produce the desired number of each type of world. This set of worlds is then saved and used by each contestant's program. If the tournament program is started with a pre-existing set of worlds, this step is skipped.

Compilation

Each contestant's source code is then compiled. This means having an executable produced for the C++ agents, a jar file for the java agents, and python bytecode files for the python agents. Any error during this stage results in a disqualification and a report detailed in the contestant's meta file, which will include the compilation errors.

Execution

During the final step of execution, each submission will be executed as its own process against the set of worlds generated earlier. There will be a timeout feature implemented, which will disqualify any agent lingering longer than a predefined number of minutes and log an appropriate message. If an agent crashes on at least one world, this contestant will also be disqualified. Scores for each contestant will be produced and continued on to the output preparation stage.

Output Preparation

The Output Preparation stage is in charge of organizing results, packaging tournament files, and verifying contestants did not cheat. This is accomplished by collecting results, writing a teacher's spreadsheet, writing a contestant scoreboard, verifying results, sorting results, and packaging output.

Result Collection

In this step, all result files are read and scores are written to the contestant's meta file. Each meta file should be saturated with information now.

Scoreboard Creation

A scoreboard is constructed for the contestants detailing their team name, average score, standard deviation, language, and any errors. In addition to this, the scoreboard will record the reason for disqualification, if there was one. This scoreboard is created by reading each contestant's meta file and skipping over confidential information.

Spreadsheet Creation

A spreadsheet is constructed for the tournament instructors detailing all information, which includes contestant's UCI Net ID, last name, student ID, team name, submission name, average score, standard deviation, language, and any errors. This spreadsheet also details disqualification reasons for those that were disqualified. A hanging column is left blank for verification stage. This spreadsheet is created by reading each contestant's meta file.

Verification

This step checks the spreadsheet for suspicious information and flags any questionable results in the hanging column made by the previous step. A result may come into question if its score matches the score of another result or if its standard deviation is unusually low. A low standard deviation might mean that agent followed the same instructions for every world.

Sorter

The final step of the tournament sorts both the scoreboard and spreadsheet by descending score.

Output Packaging

After all is said and done, all desired output files are collected and organized into the output folder.

VIII. Threat Model

Threats

i. Students who are able to mess with other students score during the tournament process
ii. Students who are able to cheat by messing with the scoring system
iii. Students who are able to break the tournament program
iv. Students spoofing their identity to falsely be on a winning team
v. Students coping the program of another student
vi. Students who turn in too simple of an agent

Countermeasures

1. The tournament program runs contestant's program in their own process. This measure makes sure each contestant gets treated equally and fairly. At the end of one contestant's process, everything is brought back to a clean-slate state. The tournament process encapsulates each student's submission into an individual program. This measure is taken to prevent (i) and (iii).
2. If a student's program breaks, by error, exception, or crash, this error will be recorded and the tournament will move on. Since every submission is encapsulated in its own process, one fatal error won't break the tournament program. This measure is taken to prevent (iii).
3. Students are confined to code only in the MyAI class. They have access to the source code for the World class, which is responsible for creating the environment, running the agent, keeping track of agent variables, and producing a score. These functions and variables are

secure under the private modifier, so from the MyAI class modifying world values cannot occur. The tournament process will extract the MyAI class source code from the student's submission and compile it with a clean World class. If the student had made malicious changes to the World class, these changes will be wiped when the MyAI class is compiled with a clean shell. This measure prevents (ii).

4. The tournament program's output will have one line for every submission. If a student tried to spoof their identity, it wouldn't matter. The student in question will have his or her line contain the scores and data he or she received. This measure prevents (iv).
5. The tournament program will run all agent submissions on the same set of worlds. If two students decided to cheat, their score will be the same. The tournament program will identify this and flag their lines in the output. This measure prevents (v).
6. Any agent that acts in a very simple, repetitive, way, will have this reflected in their standard deviations. The tournament program will identify this during the flag check state. This measure prevents (iv).

IX. Recommended Policies

Submission Policy Recommendations

Submission Names

A student's EEE Dropbox submission should be packaged as a zip archive. The archive should be named: LASTNAME_STUDENTID#_TEAMNAME.zip**.** Team names should be a unique string of alphanumeric characters. Team names with underscores, spaces, or other special unix characters will cause problems during the tournament.

Using zip archives allows the scripts to effortlessly extract submissions. Additionally, the suggested naming scheme makes it easy to collect important information; if extra underscores are not allowed, the scripts can simply treat underscores as delimiters.

Any changes to these policies will merit major changes to the tournament, some portions of the shells, and the student booklet. The shell makefile command, make submission, would need to be updated. The validation stage of the tournament would need to be updated.

Submission Requirements

The directory structure of the archive shouldn't matter, however, a student's submission archive is required to contain the MyAI class files. A pdf document should be required as well. The other source files are not necessary because the tournament will provide clean versions.

After extracting the archive, the tournament will search for the MyAI class files and a pdf document. Any extra files, beyond the required ones, will be ignored by the tournament program. Any change here will require an update to the verification step in the tournament.

Tournament Policy Recommendations

Timeout Value

An agent shouldn't take longer than thirty minutes to complete an absurd amount of worlds. With that said, if grading is based primarily on performance measure, every agent should have a fair opportunity at completing their task. The default timeout value is two hours; this can be changed inside the tournament's makefile script. Two hours works great because every agent will be run simultaneously. This sets an upper-bound of around two and a half hours for the whole tournament process.

World Set

A set of ten thousand worlds should be plenty to get a good sense of how well an agent can think. The tournament uses twenty-five hundred 4x4 worlds and five hundred of each world size up to 7x7. This gives a total of ten thousand. Any change here would require an update in the world generation portion of the tournament as well as the makefile script of the world generator.

Recommended Timeline

Three deadlines accompanied by three tournaments allows the students ample time to see how they fair up against the rest of the class and to recover if necessary. It is recommended that three tournaments are held for the students throughout the course.

First Deadline

The First Deadline tournament should focus primarily on everyone, including the instructors, getting set up. If this deadline comes really early, all students will feel comfortable working in the environment of the project when the more serious deadlines roll around. No important class topics need to be discussed before this deadline. A score of greater than or equal to -200 should be expected, as this is a relatively easy result.

Second Deadline

The Second Deadline should let the students know how they fair up against everyone else in their class. This deadline should come close to the final deadline as it serves as an almost done test. The time between this deadline and the next is when competitive students will be fine tuning their agents. The search unit will really help students with this and the next deadline. A score of greater than or equal to 1 should be expected, as this shows a positive average. This indicates that the student's agent is able to grab the gold and not die more often than not.

Final Deadline

The Final Deadline should be close to the second deadline, and it should focus on the final result of the student's project. A score of greater than 200 is entirely possible, and shows the agent is able to grab the gold in a good portion of the worlds.

X. Test Cases

Several tests were administered in an effort to minimize bugs in the final release. The following tests were conducted on UCI's Openlab with the most up-to-date version of the shells.

Shells

Several important features of the shells are tested.

Makefile

These tests correspond to the makefile script included with all shells. The script contains two commands, **make** and **make submission**. The former compiles the shell; the latter prompts the user for their last name, student ID, and team name, then packages the shell into a properly named zip-file ready for submission. These tests are operated using blank shells and the default openlab modules.

C++

Test	Expected Result	Observed Result
make	A compiled executable named "Wumpus_World" is created inside the bin folder. No output printed to the console.	A compiled executable named "Wumpus_World" is created inside the bin folder. No output printed to the console.
make submission	First, the shell is compiled using the make command. Afterward, the console asks for your last name, student ID, and team name. It then packages the bin, src, and doc folders into a zip file named according to the policies.	First, the shell is compiled using the make command. Afterward, the console asks for your last name, student ID, and team name. It then packages the bin, src, and doc folders into a zip file named according to the policies.
make submission (A second time with the same input)	An updated zip file is made without error.	An updated zip file is made without error.
make dummy	An error message stating no dummy command. No change.	"make: *** No rule to make target `dummy'. Stop." No change.

Java

Test	Expected Result	Observed Result

make	A compiled jar file named "Wumpus_World" is created inside the bin folder. No output printed to the console.	A compiled jar file named "Wumpus_World" is created inside the bin folder. No output printed to the console.
make submission	First, the shell is compiled using the make command. Afterward, the console asks for your last name, student ID, and team name. It then packages the bin, src, and doc folders into a zip file named according to the policies.	First, the shell is compiled using the make command. Afterward, the console asks for your last name, student ID, and team name. It then packages the bin, src, and doc folders into a zip file named according to the policies.
make submission (A second time with the same input)	An updated zip file is made without error.	An updated zip file is made without error.
make dummy	An error message stating no dummy command. No change.	"make: *** No rule to make target `dummy'. Stop." No change.

Python

Test	**Expected Result**	**Observed Result**
make	Compiled python bytecode files created inside the bin folder. No output printed to the console.	Compiled python bytecode files created inside the bin folder. No output printed to the console.
make submission	First, the shell is compiled using the make command. Afterward, the console asks for your last name, student ID, and team name. It then packages the bin, src, and doc folders into a zip file named according to the policies.	First, the shell is compiled using the make command. Afterward, the console asks for your last name, student ID, and team name. It then packages the bin, src, and doc folders into a zip file named according to the policies.
make submission (A second time with the same input)	An updated zip file is made without error.	An updated zip file is made without error.

make dummy	An error message stating no dummy command. No change.	"make: *** No rule to make target `dummy'. Stop." No change.

ManualAI

When the -m option is specified, the shell uses the ManualAI instead of the other agents. The ManualAI asks the user for an action at every turn, which gives full control to the omniscient player. These tests correspond to user input, where the domain is all possible keyboard input. The ManualAI should ignore all blank space up to the first character, and ignore everything after the first character. If the agent reads an 'a' it should turn left; if the agent reads a 'd' it should turn right; if the agent reads a 'w' it should move forward; if the agent reads a 's' it should shoot the arrow; if the agent reads a 'g' it should grab. All other valid characters should make the agent climb. These tests are performed on blank shells with the "python/3.5.2" openlab module loaded.

C++

Test	**Expected Result**	**Observed Result**
a	Turn left	Turn left
d	Turn right	Turn right
w	Move forward	Move forward
s	Shoot the arrow	Shoot the arrow
g	Grab	Grab
c	Climb	Climb
x	Climb	Climb
adwsgcx	Turn left; next turn behaves normally	Turn left; next turn behaves normally
'tab' a	Turn left	Turn left
'tab' a 'tab' w	Turn left; next turn behaves normally	Turn left; next turn behaves normally
'newline' a 'tab'	Turn left	Turn left

Java

Test	**Expected Result**	**Observed Result**
a	Turn left	Turn left
d	Turn right	Turn right
w	Move forward	Move forward
s	Shoot the arrow	Shoot the arrow
g	Grab	Grab
c	Climb	Climb
x	Climb	Climb

adwsgcx	Turn left; next turn behaves normally	Turn left; next turn behaves normally
‘tab’ a	Turn left	Turn left
‘tab’ a ‘tab’ w	Turn left; next turn behaves normally	Turn left; next turn behaves normally
‘newline’ a ‘tab’	Turn left	Turn left

Python

Test	Expected Result	Observed Result
a	Turn left	Turn left
d	Turn right	Turn right
w	Move forward	Move forward
s	Shoot the arrow	Shoot the arrow
g	Grab	Grab
c	Climb	Climb
x	Climb	Climb
adwsgcx	Turn left; next turn behaves normally	Turn left; next turn behaves normally
‘tab’ a	Turn left	Turn left
‘tab’ a ‘tab’ w	Turn left; next turn behaves normally	Turn left; next turn behaves normally
‘newline’ a ‘tab’	Turn left	Turn left

Command Line and Main

The main file is the entry point for the program and serves as the interface between the shell and the command line. This portion of the code will parse the command line options and construct the required amount of worlds with the correct options. These tests correspond to the main file; therefore, these tests assume than any file read is a valid world file. Since these tests focus on command line parsing, these tests assume that the board is constructed properly; board construction will be tested in the world creation section. The domain is all command line inputs. The expected behavior is described by the Shell Manual. These tests are performed on blank shells with “python/3.5.2” openlab module loaded.

C++

Test	Expected Result	Observed Result
Wumpus_World	A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.

Wumpus_World -m	A random world is constructed; the ManualAI agent is initialized, and the final score is printed to the console.	A random world is constructed; the ManualAI agent is initialized, and the final score is printed to the console.
Wumpus_World -d	A random world is constructed with the debug flag; the MyAI agent is initialized, and the final score is printed to the console.	A random world is constructed with the debug flag; the MyAI agent is initialized, and the final score is printed to the console.
Wumpus_World -r	A random world is constructed; the RandomAI agent is initialized, and the final score is printed to the console.	A random world is constructed; the RandomAI agent is initialized, and the final score is printed to the console.
Wumpus_World -h	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
Wumpus_World -rd	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.
Wumpus_World world.txt (world.txt exists and is valid)	The world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	The world is constructed; the MyAI agent is initialized, and the final score is printed to the console.
Wumpus_World -f worlds/ (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; on every world, the MyAI agent is initialized, and the average score and standard deviation are printed to the console.	The same number of worlds in the specified folder are constructed; on every world, the MyAI agent is initialized, and the average score and standard deviation are printed to the console.
Wumpus_World -rf worlds/ (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation is printed to the console.	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation is printed to the console.
Wumpus_World world.txt output.txt	The world is constructed; the MyAI agent is initialized, and	The world is constructed; the MyAI agent is initialized, and

(world.txt exists and is valid)	the final score is printed to output.txt.	the final score is printed to output.txt.
Wumpus_World -rf worlds/ output.txt (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.
Wumpus_World -rvf worlds/ output.txt (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; world names are printed to the console before loading; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.	The same number of worlds in the specified folder are constructed; world names are printed to the console before loading; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.
Wumpus_World -f	A warning message stating no valid folder found is printed to the console. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	A warning message stating no valid folder found is printed to the console. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.
Wumpus_World -f world.txt (world.txt exists and is valid)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
Wumpus_World -mdrhf	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
Wumpus_World -h world.txt	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
Wumpus_World -mr	A warning message stating two agents specified is printed to the console. ManualAI is turned off. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	A warning message stating two agents specified is printed to the console. ManualAI is turned off. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.

Wumpus_World world.txt (world.txt does not exist)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
Wumpus_World -r world.txt (world.txt does not exist)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
Wumpus_World -f worlds/ (worlds/ does not exist)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
Wumpus_World -xyz	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
Wumpus_World -RD	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.
Wumpus_World world.txt -r (world.txt exists and is valid)	The world is constructed; the MyAI agent is initialized, and the final score is printed to a file called '-r'.	The world is constructed; the MyAI agent is initialized, and the final score is printed to a file called '-r'.

Java

Test	**Expected Result**	**Observed Result**
java -jar Wumpus_World.jar	A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.
java -jar Wumpus_World.jar -m	A random world is constructed; the ManualAI agent is initialized, and the final score is printed to the console.	A random world is constructed; the ManualAI agent is initialized, and the final score is printed to the console.
java -jar Wumpus_World.jar -d	A random world is constructed with the debug flag; the MyAI agent is initialized, and the final score is printed to the console.	A random world is constructed with the debug flag; the MyAI agent is initialized, and the final score is printed to the console.
java -jar Wumpus_World.jar -r	A random world is constructed; the RandomAI	A random world is constructed; the RandomAI

	agent is initialized, and the final score is printed to the console.	agent is initialized, and the final score is printed to the console.
java -jar Wumpus_World.jar -h	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
java -jar Wumpus_World.jar -rd	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.
java -jar Wumpus_World.jar world.txt (world.txt exists and is valid)	The world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	The world is constructed; the MyAI agent is initialized, and the final score is printed to the console.
java -jar Wumpus_World.jar -f worlds/ (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; on every world, the MyAI agent is initialized, and the average score and standard deviation are printed to the console.	The same number of worlds in the specified folder are constructed; on every world, the MyAI agent is initialized, and the average score and standard deviation are printed to the console.
java -jar Wumpus_World.jar -rf worlds/ (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation is printed to the console.	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation is printed to the console.
java -jar Wumpus_World.jar world.txt output.txt (world.txt exists and is valid)	The world is constructed; the MyAI agent is initialized, and the final score is printed to output.txt.	The world is constructed; the MyAI agent is initialized, and the final score is printed to output.txt.
java -jar Wumpus_World.jar -rf worlds/ output.txt (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.

	deviation are printed to output.txt.	
java -jar Wumpus_World.jar -rvf worlds/ output.txt (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; world names are printed to the console before loading; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.	The same number of worlds in the specified folder are constructed; world names are printed to the console before loading; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.
java -jar Wumpus_World.jar -f	A warning message stating no valid folder found is printed to the console. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	A warning message stating no valid folder found is printed to the console. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.
java -jar Wumpus_World.jar -f world.txt (world.txt exists and is valid)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
java -jar Wumpus_World.jar -mdrhf	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
java -jar Wumpus_World.jar -h world.txt	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
java -jar Wumpus_World.jar -mr	A warning message stating two agents specified is printed to the console. ManualAI is turned off. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	A warning message stating two agents specified is printed to the console. ManualAI is turned off. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.
java -jar Wumpus_World.jar world.txt (world.txt does not exist)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.

java -jar Wumpus_World.jar -r world.txt (world.txt does not exist)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
java -jar Wumpus_World.jar -f worlds/ (worlds/ does not exist)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
java -jar Wumpus_World.jar -xyz	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
java -jar Wumpus_World.jar -RD	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.
java -jar Wumpus_World.jar world.txt -r (world.txt exists and is valid)	The world is constructed; the MyAI agent is initialized, and the final score is printed to a file called '-r'.	The world is constructed; the MyAI agent is initialized, and the final score is printed to a file called '-r'.

Python

Test	**Expected Result**	**Observed Result**
python3 Main.pyc	A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.
python3 Main.pyc -m	A random world is constructed; the ManualAI agent is initialized, and the final score is printed to the console.	A random world is constructed; the ManualAI agent is initialized, and the final score is printed to the console.
python3 Main.pyc -d	A random world is constructed with the debug flag; the MyAI agent is initialized, and the final score is printed to the console.	A random world is constructed with the debug flag; the MyAI agent is initialized, and the final score is printed to the console.
python3 Main.pyc -r	A random world is constructed; the RandomAI agent is initialized, and the	A random world is constructed; the RandomAI agent is initialized, and the

	final score is printed to the console.	final score is printed to the console.
python3 Main.pyc -h	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
python3 Main.pyc -rd	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.
python3 Main.pyc world.txt (world.txt exists and is valid)	The world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	The world is constructed; the MyAI agent is initialized, and the final score is printed to the console.
python3 Main.pyc -f worlds/ (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; on every world, the MyAI agent is initialized, and the average score and standard deviation are printed to the console.	The same number of worlds in the specified folder are constructed; on every world, the MyAI agent is initialized, and the average score and standard deviation are printed to the console.
python3 Main.pyc -rf worlds/ (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation is printed to the console.	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation is printed to the console.
python3 Main.pyc world.txt output.txt (world.txt exists and is valid)	The world is constructed; the MyAI agent is initialized, and the final score is printed to output.txt.	The world is constructed; the MyAI agent is initialized, and the final score is printed to output.txt.
python3 Main.pyc -rf worlds/ output.txt (worlds/ exists and contains multiple valid world files)	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.	The same number of worlds in the specified folder are constructed; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.
python3 Main.pyc -rvf worlds/ output.txt	The same number of worlds in the specified folder are constructed; world names are printed to the console	The same number of worlds in the specified folder are constructed; world names are printed to the console

(worlds/ exists and contains multiple valid world files)	before loading; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.	before loading; on every world, the RandomAI agent is initialized, and the average score and standard deviation are printed to output.txt.
python3 Main.pyc -f	A warning message stating no valid folder found is printed to the console. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	A warning message stating no valid folder found is printed to the console. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.
python3 Main.pyc -f world.txt (world.txt exists and is valid)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
python3 Main.pyc -mdrhf	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
python3 Main.pyc -h world.txt	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.
python3 Main.pyc -mr	A warning message stating two agents specified is printed to the console. ManualAI is turned off. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.	A warning message stating two agents specified is printed to the console. ManualAI is turned off. A random world is constructed; the MyAI agent is initialized, and the final score is printed to the console.
python3 Main.pyc world.txt (world.txt does not exist)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
python3 Main.pyc -r world.txt (world.txt does not exist)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
python3 Main.pyc -f worlds/ (worlds/ does not exist)	An error message is printed to the console. The program terminates.	An error message is printed to the console. The program terminates.
python3 Main.pyc -xyz	The help menu is printed. The program then terminates.	The help menu is printed. The program then terminates.

python3 Main.pyc -RD	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.	A random world is constructed with the debug flag; the RandomAI agent is initialized, and the final score is printed to the console.
python3 Main.pyc world.txt -r (world.txt exists and is valid)	The world is constructed; the MyAI agent is initialized, and the final score is printed to a file called '-r'.	The world is constructed; the MyAI agent is initialized, and the final score is printed to a file called '-r'.

World Creation

These tests check that the correct world is created when loading a file. They also check that if an invalid file is read, an appropriate error is issued. The world test set contains both UNIX and DOS files. These tests are performed on blank shells with "python/3.5.2" openlab module loaded.

C++

Test	**Expected Result**	**Observed Result**
Loading a valid world file	The board is constructed as specified in the world file.	The board is constructed as specified in the world file.
Loading an invalid world file	An error message is printed to the console.	An error message is printed to the console.
Loading a blank file	An error message is printed to the console.	An error message is printed to the console.
Loading a folder of valid world files	All of the worlds are constructed correctly.	All of the worlds are constructed correctly.
Loading an empty folder	Null average score and standard deviation are printed to the console.	Null average score and standard deviation are printed to the console.
Loading a folder of mostly valid world files	Null average score and standard deviation are printed to the console.	Null average score and standard deviation are printed to the console.
Loading a folder of all invalid world files	Null average score and standard deviation are printed to the console.	Null average score and standard deviation are printed to the console.

Java

Test	**Expected Result**	**Observed Result**
Loading a valid world file	The board is constructed as specified in the world file.	The board is constructed as specified in the world file.

Loading an invalid world file	An error message is printed to the console.	An error message is printed to the console.
Loading a blank file	An error message is printed to the console.	An error message is printed to the console.
Loading a folder of valid world files	All of the worlds are constructed correctly.	All of the worlds are constructed correctly.
Loading an empty folder	Null average score and standard deviation are printed to the console.	Null average score and standard deviation are printed to the console.
Loading a folder of mostly valid world files	Null average score and standard deviation are printed to the console.	Null average score and standard deviation are printed to the console.
Loading a folder of all invalid world files	Null average score and standard deviation are printed to the console.	Null average score and standard deviation are printed to the console.

Python

Test	**Expected Result**	**Observed Result**
Loading a valid world file	The board is constructed as specified in the world file.	The board is constructed as specified in the world file.
Loading an invalid world file	An error message is printed to the console.	An error message is printed to the console.
Loading a blank file	An error message is printed to the console.	An error message is printed to the console.
Loading a folder of valid world files	All of the worlds are constructed correctly.	All of the worlds are constructed correctly.
Loading an empty folder	Null average score and standard deviation are printed to the console.	Null average score and standard deviation are printed to the console.
Loading a folder of mostly valid world files	Null average score and standard deviation are printed to the console.	Null average score and standard deviation are printed to the console.
Loading a folder of all invalid world files	Null average score and standard deviation are printed to the console.	Null average score and standard deviation are printed to the console.

Game Mechanics

These tests check that the game mechanics are implemented correctly. The ManualAI is used for a bulk portion of these tests. These tests are performed on blank shells with "python/3.5.2" openlab module loaded.

C++

Test	Expected Result	Observed Result
Grab when not in room with gold	Nothing. -1 point.	Nothing. -1 point.
Grab when in room with gold.	Gold disappears off board. Agent now has gold. -1 point.	Gold disappears off board. Agent now has gold. -1 point.
Climb from (1, 1)	-1 point. Game ends.	-1 point. Game ends.
Climb from (1, 1) with gold	-1 point. +1000 points, game ends.	-1 point. +1000 points, game ends.
Climb from anywhere else	Nothing. -1 point.	Nothing. -1 point.
Shoot when Wumpus in line of sight and have arrow	Wumpus disappears off board; a stench appears in the same spot. A scream is perceived. Agent doesn't have an arrow anymore. -11 points.	Wumpus disappears off board; a stench appears in the same spot. A scream is perceived. Agent doesn't have an arrow anymore. -11 points.
Shoot when Wumpus in line of sight and do not have arrow	Nothing. -1 point.	Nothing. -1 point.
Shoot when Wumpus out of sight and have arrow	Nothing. -11 points.	Nothing. -11 points.
Turn left when facing right	Face up. -1 point.	Face up. -1 point.
Turn left when facing up	Face left. -1 point.	Face left. -1 point.
Turn left when facing left	Face down. -1 point.	Face down. -1 point.
Turn left when facing down	Face right. -1 point.	Face right. -1 point.
Turn right when facing right	Face down. -1 point.	Face down. -1 point.
Turn right when facing up	Face right. -1 point.	Face right. -1 point.
Turn right when facing left	Face up. -1 point.	Face up. -1 point.
Turn right when facing down	Face left. -1 point.	Face left. -1 point.
Move forward	Move forward. -1 point.	Move forward. -1 point.
Move forward into the Wumpus	Die. -1001 points.	Die. -1001 points.
Move forward into a pit	Die. -1001 points.	Die. -1001 points.
Move forward into top wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
Move forward into left wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
Move forward into right wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
Move forward into bottom wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
In a room with a breeze	Perceive a breeze.	Perceive a breeze.
In a room without a breeze	Nothing.	Nothing.
In a room with a stench	Perceive a stench.	Perceive a stench.
In a room without a stench	Nothing.	Nothing.

Java

Test	Expected Result	Observed Result
Grab when not in room with gold	Nothing. -1 point.	Nothing. -1 point.
Grab when in room with gold.	Gold disappears off board. Agent now has gold. -1 point.	Gold disappears off board. Agent now has gold. -1 point.
Climb from (1, 1)	-1 point. Game ends.	-1 point. Game ends.
Climb from (1, 1) with gold	-1 point. +1000 points, game ends.	-1 point. +1000 points, game ends.
Climb from anywhere else	Nothing. -1 point.	Nothing. -1 point.
Shoot when Wumpus in line of sight and have arrow	Wumpus disappears off board; a stench appears in the same spot. A scream is perceived. Agent doesn't have an arrow anymore. -11 points.	Wumpus disappears off board; a stench appears in the same spot. A scream is perceived. Agent doesn't have an arrow anymore. -11 points.
Shoot when Wumpus in line of sight and do not have arrow	Nothing. -1 point.	Nothing. -1 point.
Shoot when Wumpus out of sight and have arrow	Nothing. -11 points.	Nothing. -11 points.
Turn left when facing right	Face up. -1 point.	Face up. -1 point.
Turn left when facing up	Face left. -1 point.	Face left. -1 point.
Turn left when facing left	Face down. -1 point.	Face down. -1 point.
Turn left when facing down	Face right. -1 point.	Face right. -1 point.
Turn right when facing right	Face down. -1 point.	Face down. -1 point.
Turn right when facing up	Face right. -1 point.	Face right. -1 point.
Turn right when facing left	Face up. -1 point.	Face up. -1 point.
Turn right when facing down	Face left. -1 point.	Face left. -1 point.
Move forward	Move forward. -1 point.	Move forward. -1 point.
Move forward into the Wumpus	Die. -1001 points.	Die. -1001 points.
Move forward into a pit	Die. -1001 points.	Die. -1001 points.
Move forward into top wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
Move forward into left wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
Move forward into right wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
Move forward into bottom wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
In a room with a breeze	Perceive a breeze.	Perceive a breeze.
In a room without a breeze	Nothing.	Nothing.

In a room with a stench	Perceive a stench.	Perceive a stench.
In a room without a stench	Nothing.	Nothing.

Python

Test	**Expected Result**	**Observed Result**
Grab when not in room with gold	Nothing. -1 point.	Nothing. -1 point.
Grab when in room with gold.	Gold disappears off board. Agent now has gold. -1 point.	Gold disappears off board. Agent now has gold. -1 point.
Climb from (1, 1)	-1 point. Game ends.	-1 point. Game ends.
Climb from (1, 1) with gold	-1 point. +1000 points, game ends.	-1 point. +1000 points, game ends.
Climb from anywhere else	Nothing. -1 point.	Nothing. -1 point.
Shoot when Wumpus in line of sight and have arrow	Wumpus disappears off board; a stench appears in the same spot. A scream is perceived. Agent doesn't have an arrow anymore. -11 points.	Wumpus disappears off board; a stench appears in the same spot. A scream is perceived. Agent doesn't have an arrow anymore. -11 points.
Shoot when Wumpus in line of sight and do not have arrow	Nothing. -1 point.	Nothing. -1 point.
Shoot when Wumpus out of sight and have arrow	Nothing. -11 points.	Nothing. -11 points.
Turn left when facing right	Face up. -1 point.	Face up. -1 point.
Turn left when facing up	Face left. -1 point.	Face left. -1 point.
Turn left when facing left	Face down. -1 point.	Face down. -1 point.
Turn left when facing down	Face right. -1 point.	Face right. -1 point.
Turn right when facing right	Face down. -1 point.	Face down. -1 point.
Turn right when facing up	Face right. -1 point.	Face right. -1 point.
Turn right when facing left	Face up. -1 point.	Face up. -1 point.
Turn right when facing down	Face left. -1 point.	Face left. -1 point.
Move forward	Move forward. -1 point.	Move forward. -1 point.
Move forward into the Wumpus	Die. -1001 points.	Die. -1001 points.
Move forward into a pit	Die. -1001 points.	Die. -1001 points.
Move forward into top wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
Move forward into left wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
Move forward into right wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.
Move forward into bottom wall	Perceive a bump. -1 point.	Perceive a bump. -1 point.

In a room with a breeze	Perceive a breeze.	Perceive a breeze.
In a room without a breeze	Nothing.	Nothing.
In a room with a stench	Perceive a stench.	Perceive a stench.
In a room without a stench	Nothing.	Nothing.

World Generator

These tests will ensure the world generator generates only valid worlds; it will also check that the correct amount of worlds were created. The first set of tests focus on the execution of the Binary, while the second set of tests focus on the makefile script. The following tests were performed on UCI's openlab with the default modules loaded.

Test	**Expected Result**	**Observed Result**
World_Generator world 1000 4 4	1000 valid 4x4 worlds are generated.	1000 valid 4x4 worlds are generated.
World_Generator world 1000 4 5	1000 valid 4x5 worlds are generated.	1000 valid 4x5 worlds are generated.
World_Generator world 1000 5 4	1000 valid 5x4 worlds are generated.	1000 valid 5x4 worlds are generated.
World_Generator world 1000	An error message detailing the correct format is printed to the console.	An error message detailing the correct format is printed to the console.

Test	**Expected Result**	**Observed Result**
make	The console prompts the user for row dimension, column dimension, and world count. The desired set of worlds is created in the Worlds folder.	The console prompts the user for row dimension, column dimension, and world count. The desired set of worlds is created in the Worlds folder.
make tournamentSet	The tournament set of worlds is created in the Worlds folder.	The tournament set of worlds is created in the Worlds folder.
make dummy	An error message stating no dummy command. No change.	"make: *** No rule to make target `dummy'. Stop." No change.

Tournament

Input Preparation

These tests check that all parts of the Input Preparation are operating correctly. Each test corresponds to one portion of this part of the tournament. The domain of a test is the successes of the previous test.

Input Collection

Test	**Expected Result**	**Observed Result**
A single student submission	The submission is copied into a temporary submission folder.	The submission is copied into a temporary submission folder.
A single EEE archive with many submissions inside	All submissions inside the EEE archive are extracted into a temporary submission folder.	All submissions inside the EEE archive are extracted into a temporary submission folder.
A mixture of student submissions and EEE archives	All submissions inside the EEE archive are extracted into a temporary submission folder. All submissions outside the EEE archive are extracted into a temporary submission folder.	All submissions inside the EEE archive are extracted into a temporary submission folder. All submissions outside the EEE archive are extracted into a temporary submission folder.

Meta Folder Creation

Test	**Expected Result**	**Observed Result**
Student Submission with incorrect name	A meta directory is created. A subfolder with the submission name is created. The submission is placed inside the folder and a meta file is created.	A meta directory is created. A subfolder with the submission name is created. The submission is placed inside the folder and a meta file is created.
Student Submission with correct name	A meta directory is created. A subfolder with the student's UCINetID is created. The submission is placed inside the folder and a meta file is created.	A meta directory is created. A subfolder with the student's UCINetID is created. The submission is placed inside the folder and a meta file is created.

Validation

Test	**Expected Result**	**Observed Result**

Correct Student Submission	The Meta file is updated with UCINetID, Last name, student ID, and team name.	The Meta file is updated with UCINetID, Last name, student ID, and team name.
Student Submission not in zip format	The Meta file is updated with a fatal error and the submission is disqualified.	The Meta file is updated with a fatal error and the submission is disqualified.
Student Submission with incorrect name	The Meta file is updated with a fatal error and the submission is disqualified.	The Meta file is updated with a fatal error and the submission is disqualified.
Student Submission not in zip format and with incorrect name	The Meta file is updated with both fatal errors and the submission is disqualified.	The Meta file is updated with both fatal errors and the submission is disqualified.

Extraction

Test	**Expected Result**	**Observed Result**
Correct Student Submission Archive	The student submission archive is extracted to a folder inside of the students working directory.	The student submission archive is extracted to a folder inside of the students working directory.
Corrupted Student Submission Archive	The Meta file is updated with a fatal error and the submission is disqualified.	The Meta file is updated with a fatal error and the submission is disqualified.

Verification

Test	**Expected Result**	**Observed Result**
Student Submission with pdf document and Python source code	The pdf file is extracted to a document folder. The MyAI source class is extracted to a source folder. The Meta file is updated with language information.	The pdf file is extracted to a document folder. The MyAI source class is extracted to a source folder. The Meta file is updated with language information.
Student Submission with pdf document and Java source code	The pdf file is extracted to a document folder. The MyAI source class is extracted to a source folder. The Meta file is updated with language information.	The pdf file is extracted to a document folder. The MyAI source class is extracted to a source folder. The Meta file is updated with language information.
Student Submission with pdf document and C++ source code	The pdf file is extracted to a document folder. The MyAI source class is extracted to a source folder. The Meta file	The pdf file is extracted to a document folder. The MyAI source class is extracted to a source folder. The Meta file

	is updated with language information.	is updated with language information.
Student Submission with pdf document and no source code	The pdf file is extracted to a document folder. The Meta file is updated with a fatal error and the submission is disqualified.	The pdf file is extracted to a document folder. The Meta file is updated with a fatal error and the submission is disqualified.
Student Submission with Python source code and no pdf document	The Meta file is updated with an error. The MyAI source class is extracted to a source folder. The Meta file is updated with language information.	The Meta file is updated with an error. The MyAI source class is extracted to a source folder. The Meta file is updated with language information.
Student Submission with Java source code and no pdf document	The Meta file is updated with an error. The MyAI source class is extracted to a source folder. The Meta file is updated with language information.	The Meta file is updated with an error. The MyAI source class is extracted to a source folder. The Meta file is updated with language information.
Student Submission with C++ source code and no pdf document	The Meta file is updated with an error. The MyAI source class is extracted to a source folder. The Meta file is updated with language information.	The Meta file is updated with an error. The MyAI source class is extracted to a source folder. The Meta file is updated with language information.
Student Submission with no Source code and no pdf document	The Meta file is updated with an error. The Meta file is updated with a fatal error and the submission is disqualified.	The Meta file is updated with an error. The Meta file is updated with a fatal error and the submission is disqualified.

Compilation Preparation

Test	**Expected Result**	**Observed Result**
Student Submission with Python source code	A blank Python shell is copied into the student's source folder.	A blank Python shell is copied into the student's source folder.
Student Submission with Java source code	A blank Java shell is copied into the student's source folder.	A blank Java shell is copied into the student's source folder.
Student Submission with C++ source code	A blank C++ shell is copied into the student's source folder.	A blank C++ shell is copied into the student's source folder.

Execution

These tests make sure that valid and verified contestants are executed correctly, scores are reported correctly, and worlds are generated correctly.

World Generation

Test	Expected Result	Observed Result
Worlds are not supplied by tournament administrator	10000 worlds of variable size from 4x4 to 7x7 are created and placed inside the world folder.	10000 worlds of variable size from 4x4 to 7x7 are created and placed inside the world folder.
Worlds are supplied by tournament administrator	Nothing is created.	Nothing is created.

Compilation

Test	Expected Result	Observed Result
A contestant with Python source code and no errors	The source code is compiled and python bytecode files are placed inside the bin folder.	The source code is compiled and python bytecode files are placed inside the bin folder.
A contestant with Java source code and no errors	The source code is compiled and a jar file is placed inside the bin folder.	The source code is compiled and a jar file is placed inside the bin folder.
A contestant with C++ source code and no errors	The source code is compiled and an executable file is placed inside the bin folder.	The source code is compiled and an executable file is placed inside the bin folder.
A contestant with Python source code and errors	A fatal error is recorded in the Meta file and the contestant is disqualified.	A fatal error is recorded in the Meta file and the contestant is disqualified.
A contestant with Java source code and errors	A fatal error is recorded in the Meta file and the contestant is disqualified.	A fatal error is recorded in the Meta file and the contestant is disqualified.
A contestant with C++ source code and errors	A fatal error is recorded in the Meta file and the contestant is disqualified.	A fatal error is recorded in the Meta file and the contestant is disqualified.

Execution

Test	Expected Result	Observed Result
A Python contestant with a smart agent	The agent is run on the 10000 worlds and scores are written to an output file.	

A Java contestant with a smart agent	The agent is run on the 10000 worlds and scores are written to an output file.	The agent is run on the 10000 worlds and scores are written to an output file.
A C++ contestant with a smart agent	The agent is run on the 10000 worlds and scores are written to an output file.	The agent is run on the 10000 worlds and scores are written to an output file.
A Python contestant with an agent that hangs	The agent is run on the 10000 worlds and times-out after the specified timeout value. A fatal error is recorded to Meta file and the contestant is disqualified.	The agent is run on the 10000 worlds and times-out after the specified timeout value. A fatal error is recorded to Meta file and the contestant is disqualified.
A Java contestant with an agent that hangs	The agent is run on the 10000 worlds and times-out after the specified timeout value. A fatal error is recorded to Meta file and the contestant is disqualified.	The agent is run on the 10000 worlds and times-out after the specified timeout value. A fatal error is recorded to Meta file and the contestant is disqualified.
A C++ contestant with an agent that hangs	The agent is run on the 10000 worlds and times-out after the specified timeout value. A fatal error is recorded to Meta file and the contestant is disqualified.	The agent is run on the 10000 worlds and times-out after the specified timeout value. A fatal error is recorded to Meta file and the contestant is disqualified.
A Python contestant with an agent that crashes	The agent is run on the 10000 worlds. When it crashes, either a NaN score is reported or no score is reported depending on how it crashed. A fatal error is recorded to Meta file and the contestant is disqualified.	The agent is run on the 10000 worlds. When it crashes, either a NaN score is reported or no score is reported depending on how it crashed. A fatal error is recorded to Meta file and the contestant is disqualified.
A Java contestant with an agent that crashes	The agent is run on the 10000 worlds. When it crashes, either a NaN score is reported or no score is reported depending on how it crashed. A fatal error is recorded to Meta file and the contestant is disqualified.	The agent is run on the 10000 worlds. When it crashes, either a NaN score is reported or no score is reported depending on how it crashed. A fatal error is recorded to Meta file and the contestant is disqualified.
A C++ contestant with an agent that crashes	The agent is run on the 10000 worlds. When it crashes, either a NaN score is	The agent is run on the 10000 worlds. When it crashes, either a NaN score is

	reported or no score is reported depending on how it crashed. A fatal error is recorded to Meta file and the contestant is disqualified.	reported or no score is reported depending on how it crashed. A fatal error is recorded to Meta file and the contestant is disqualified.

Output Preparation

These tests make sure the output is constructed correctly. They also check that the suspicious results are flagged.

Result Collection

All contestants that have a result file have their results written to their Meta file.

Scoreboard Creation

The scoreboard is created correctly from reading the Meta file.

Spreadsheet Creation

The Spreadsheet is created correctly from reading the Meta file.

Verification

Test	**Expected Result**	**Observed Result**
Two scores which are the same	One of them is flagged for suspicious behavior.	One of them is flagged for suspicious behavior.
A standard deviation less than 10	The agent is flagged for suspicious behavior.	The agent is flagged for suspicious behavior.

Sorting

Both spreadsheets are correctly sorted according to scores.

Output Collection

All output files are copied into the Output folder.