DataLog

Class Description

What are the class's goals, and what should its high-level functionality look like?

- A DataLog is an object to track a series of data values over time. It should be able to return any helpful statistics, including the mean, median, min, max, and number of values collected at any point in time. Its capabilities could be scaled up if necessary, such as setting template flags to specify what the DataLog should track.
- The goal of the DataLog is to provide us with a structure to keep the different data values generated by the world and the agents organized, and to return any useful information based on those values. This allows us to have different insights into the performance of the game by comparing any values returned by the DataLog to the values that we would be expecting to see based on our idea.

Similar Classes

What similar classes in the standard library should we be familiar with or use?

- Std::vector, std::deque,
- #include <algorithm>
- Std::chrono for timestamps in the data log.

Key Functions

What key functions are we planning on implementing?

- We definitely need a main data log function and a reset function. Add and Reset functions
- Mean, median, min, max functions
- Add to log function
- A function that displays the count

Error Conditions

What error conditions will we be responsive to? Should be indicated whether the source was programmer error, a potentially recoverable error, or a user error.

- Data type error (wrong type of data inputted) programmer
- Out of accepted range error (data inputted outside of a programmer decided range)

Expected Challenges

What challenges are we going to face with this class, and what extra topics may we need to learn about?

- The biggest challenge will be waiting on other groups to finish their main ideas so we can get started.

Other Classes

What other group's classes will we need to coordinate with?

- We will probably have to coordinate with groups 1 & 2 (the agent groups) to receive data pertaining to the agents as they take actions, such as health, location, damage dealt/taken, or any other agent attribute that we may want to keep track of and have information on.
- Additionally, the same will have to be done with groups 14 & 15 (the world groups) to receive their data pertaining to the worlds, as well as the items that are within the world.
- We will also probably have to coordinate with the GUI interface team's classes in order to display a lot of the data that we collect and store in our DataLogs.

ActionLog

Class Description

What are the class's goals, and what should its high-level functionality look like?

- An ActionLog is an object that tracks the moves made by all active agents in the current world. If necessary it should track perfect timings between moves in the event that our game relies on such information.
- The goal of the ActionLog is to be a data structure specifically designed to keep track of the moves an agent makes, that way it can be read in by a ReplayDriver and allow us to go back and look at previous runs of the program. This makes it so developers can help each other debug any issues even if they weren't there to see it when the issue occurred.

Similar Classes

What similar classes in the standard library should we be familiar with or use?

- std::vector
- std::unordered_map

Key Functions

What key functions are we planning on implementing?

- A way in tracking the movement/actions of all the active agents
- Function that creates a log for a specific action… Add/save an agent event
- Function that gives a number of current number of active agents (Same thing for events)
- Functions that log damage, deaths, picking up items, etc
- Clear Log

Error Conditions

What error conditions will we be responsive to? Should be indicated whether the source was programmer error, a potentially recoverable error, or a user error.

- Unidentified action (action not recognized or able to be stored properly)
- Out of sequence action (two actions happening in sequence that shouldn't)

Expected Challenges

What challenges are we going to face with this class, and what extra topics may we need to learn about?

- The way we implement this is heavily dependent on what sort of game is chosen. If a game with more involved/active agents is chosen, there are many more actions needed to be logged

Other Classes

What other group's classes will we need to coordinate with?

- We will definitely need to coordinate with groups 1 & 2 (agent groups) because they will provide the data pertaining to agent actions, specifically WorldPath from group 2, which tracks the movement of an agent through a world.
- Groups 14 & 15 (world groups) will also be important to coordinate with because they handle a lot of the agent movement. One key class will be StateGrid, or StateGridPosition, which tracks different positions in the state grid and manages the agents moving through them

ReplayDriver

Class Description

What are the class's goals, and what should its high-level functionality look like?

- A ReplayDriver is an object that takes control over all the agents in a world and reads in an ActionLog object to fully replay everything that occurred during a previous run of the program.
- The goal of a ReplayDriver is to allow developers to play back any runs of the program that they may need to see for debugging purposes. For example, visually being able to see where an issue occurred, therefore helping them to figure out where they need to look in the code to solve that particular issue.

Similar Classes

What similar classes in the standard library should we be familiar with or use?

- std::vector

Key Functions

What key functions are we planning on implementing?

- Function used to go through the action logs incrementally
- A way to communicate with ActionLog to display actions
- We should definitely have a function that allows us to set a seed (maybe taking each seed through a hash function). When a data log is done we send that memory somewhere (maybe only track last 10 so memory isn't an issue) (xml file?)

Error Conditions

What error conditions will we be responsive to? Should be indicated whether the source was programmer error, a potentially recoverable error, or a user error.

- Invalid action given (unable to enact the action)
- Invalid replay received (unable to perform replay from given code)

Expected Challenges

What challenges are we going to face with this class, and what extra topics may we need to learn about?

- We will need a way to communicate with ActionLog, which will be dependant on how well-implemented the interface classes between the two are
- If we are going to hash the values, we will need to correctly implement hashing

Other Classes

What other group's classes will we need to coordinate with?

- Groups 17 & 18 (interface groups) will be the most important to coordinate with when it comes to the ReplayDriver, because they are going to be the ones who need to display the replays that we will store.

Timer

Class Description

What are the class's goals, and what should its high-level functionality look like?

- A Timer is a tool used to make precise timing measurements. It should be told when to start and when to stop, but timings should each have a unique and descriptive name such that the Timer can track multiple different times and compare them.
- The goal of a Timer is to give us insight into what parts of our program are either taking longer or shorter amounts of time than intended. Additionally, if any other functions of the program require very precise timing measurements, they can use the Timer to get that information.

Similar Classes

What similar classes in the standard library should we be familiar with or use?

- There are a handful of functions from the chrono library that deal with time and duration.
- Std::chrono::time_point is a class that represents a point in time, and can be added or subtracted

Key Functions

What key functions are we planning on implementing?

- Start function, stop function (with a string name variable)
- Reset function
- Reset all timers function
- Stop function (including stopping specific timers)
- Functions to retrieve aggregates like average, max, min time

Error Conditions

What error conditions will we be responsive to? Should be indicated whether the source was programmer error, a potentially recoverable error, or a user error.

- Out of accepted bounds (time is out of a expected value set by programmer such as a negative time)

Expected Challenges

What challenges are we going to face with this class, and what extra topics may we need to learn about?

- Depending on how precise the timing needs to be, we may have challenges with rounding or off values if we're not careful on how we store timing information.
- We may face challenges translating our timing preciseness over to emscripten. There may be a little lag.
- Stop must come after a start so it should be accounted for

Other Classes

What other group's classes will we need to coordinate with?

- AI Agents
- GUI interface
- Manual agents

OutputManager

Class Description

What are the class's goals, and what should its high-level functionality look like?

- An OutputManager is a simple logging system for programmers to log events of their choice. Should have the ability to output those logs multiple ways, including Silently, where it won't output any logs at all, Normally, which will output the normal log, and Detailed, where it will output much more of the information pertaining to the event. Optionally, it will include time-stamps as well.
- The goal of an OutputManager is to give the programmer the ability to see information as the program is running, and be able to quickly notice if something is off. Additionally, they can go read through the logs following the code's execution, and if they noticed something off in the program, they can use the OutputManager to read through the logs and use them to figure out what caused the issue in the program.

Similar Classes

What similar classes in the standard library should we be familiar with or use?

- Std::ostream, std::ofstream
- Thread information most likely? Std::mutex

Key Functions

What key functions are we planning on implementing?

- Function to switch what mode you are in (debug, warn, error, info)
- Include a way to have a timestamp for a specific log

Error Conditions

What error conditions will we be responsive to? Should be indicated whether the source was programmer error, a potentially recoverable error, or a user error.

- Incorrect log type (user error where requested logs don't exist)
- Timing error (logs are not outputted in the correct sequential order)

Expected Challenges

What challenges are we going to face with this class, and what extra topics may we need to learn about?

- One challenge that we are probably going to face is how we want to display any output information, as well as exactly what information we want to collect and display.

Other Classes

What other group's classes will we need to coordinate with?

- The agent groups and the world groups (1, 2, 14, 15) will all want to make use of the OutputManager in order to output information about the events occurring and use them to debug any issues they might be facing

Module Vision

The vision for our main module is to give developers insight into player patterns such as location and gameplay stats, allowing them to alter the world and agent behavior in order to provide a better gameplay experience. Additionally, we want to be able to provide debugging information so that programmers can more easily fix bugs and solve any issues they might be having with the program.