Segfault Survivors

CLASS DESIGN REPORT

Made By Group 20 Members:

Abdulrahman
Divyalakshmi
Apar
Scott
Noyonika

DESIGN

The classes are based on assumptions of what the external classes will look like in order to handle interactions between them. The world will follow a tick timer (a global clock that resets at every run of the simulation) where each tick represents a possible agent action or world event.

The Agent class is an abstract base class. SwarmingAgent, ScriptedAgent, and ReplayAgent will all derive from this class. The derived classes are responsible for behavior/decision-making. During a live game, the SwarmingAgent and ScriptedAgent will log their action with their reference to ActionLog. ActionLog will be the interface to all of our group's classes. DataLog will be the hub of the logging classes and owns ActionLog, Output Manager, and the Timer class, which captures precise duration measurements. DataLog contains the log data. At the end of the game, DataLog will send its information to OutputManager, which outputs information to the console and saves the simulation data to a JSON file.

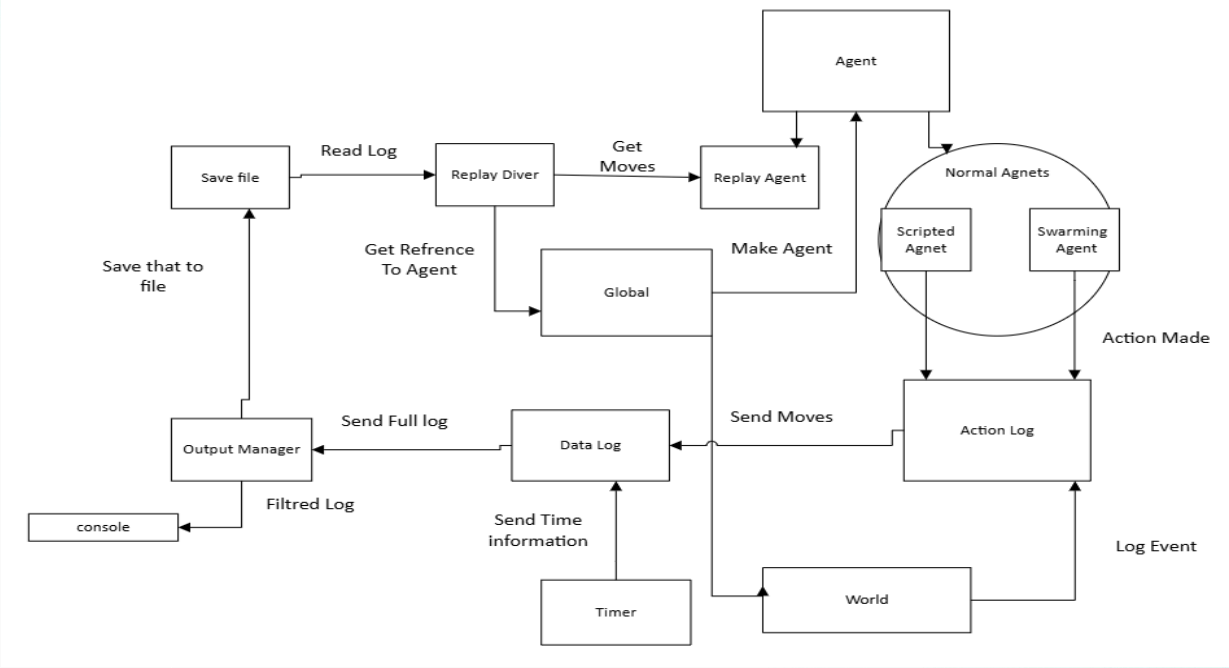

Figure 1: The Initial UML Diagram representing the classes to be developed by the Data Team for the project and how they interface with other components.

CLASSES

1. Timer()

Developed by: Abdulrahman

Description: A tool to make precise timing measurements. It should be told when to start and stop, but also be given a name for the timing so that it can track multiple different times and compare them as needed. It also captures a timestamp from the shared GlobalClock upon starting and calculates the precise difference.

Member Variables:

- name
- startTime
- endTime
- timeDifference: store the time difference (endtime - startTime)

Member Functions:

- Timer(const std::string& name):
  - Constructor to initialize the specific instance of the timer instance
- Start():
  - To capture the current start time and save it into startTime
- Stop():
  - To capture the current end time and calculate the difference from startTime
- GetTotalTime():
  - return the total recorded time
- Reset():
  - reset the member variables

Error Conditions:

- Calling GetDuration() before calling Start()

Expected Challenges:

- Ensuring that GlobalClock::Now() returns the same unit (e.g., seconds vs. milliseconds) across Windows, Mac, and Linux so that the math doesn't break when switching computers.

Similar Classes in the Standard Library:

- std::chrono::steady_clock
- std::chrono::duration

A list of any other group's C++ classes to coordinate with:

- Group 13 (Math World) & Group 19 (Interaction World)
- Group 21 (GUI) & Group 22 (Web Interface)

2. ReplayAgent()

Developed by: Abdulrahman

Description: A specialized derived class of the Agent base class. It asks the ReplayDriver for the action that the original agent performed at that specific time in the past and applies it to itself.

Member Variables:

- ReplayDriver* _driver
- Int agentId
- (inherited) double x,y

Member Functions:

- ReplayAgent(int agentID, ReplayDriver* driver)
- void Update() override

Error Conditions:

- Timeline Desync: If the simulation runs faster than the log

Expected Challenges:

- Teleportation. Eg: If the action was to move agent 5 meters, the agent would teleport to that position instead of moving smoothly
- Dependency of the Agent base class

Similar Classes in the Standard Library:

- std::istream_iterator

A list of any other group's C++ classes to coordinate with:

- Group 11 & 12 (Agent Base Class)

3. DataLog()

Developed by: Divyalakshmi

Description: DataLog is the central data aggregation and storage class for the Data Analytics group's design. It is responsible for collecting, organizing, and analyzing action-time data generated during a live simulation run.

During execution, DataLog receives validated action events from ActionLog along with precise timing information from the Timer class. Each event is recorded as a single LogEntry (a struct), which represents one agent action paired with its corresponding timestamp from Timer and metadata from ActionLog. This allows DataLog to maintain a complete chronological record of the simulation.

Member Variables:

- std::vector<LogEntry> _entries
- double _runningSum
- double _minValue
- double _maxValue
- size_t _count
- ActionLog _actionLog
    - Owned instance responsible for receiving and validating log events from agents and the world.
- OutputManager _outputManager
    - Owned instance responsible for outputting logs and statistics to console and file.

Member Functions:

- void AddEntry(const LogEntry& entry)
    - Adds a validated log entry to the DataLog and updates all running statistics.

- double GetMean() const
    - Returns the mean of the logged values.
- double GetMedian() const
    - Returns the median of the logged values.
- double GetMin() const
    - Returns the minimum recorded value.
- double GetMax() const
    - Returns the maximum recorded value.
- size_t GetCount() const
    - Returns the total number of logged entries.
- void WriteToOutput()
    - Sends the complete log and computed statistics to the OutputManager
- void Reset()
    - Clears all stored data and resets internal state for a new simulation run.

Error Conditions:

- Querying statistics when no data exists
- Malformed or incomplete log entries
- Invalid or inconsistent timing data from Timer

Expected Challenges:

- Regarding setting template flags to figure out what to track and prioritize, we are yet to explore that notion.
- We hope it's not too much responsibility given to one class. If required, we can break down this class into a couple of smaller ones.
- Based on the team and company's final decision on measuring "global" time and elapsed time (using the Timer class), our implementation of DataLog might change.
- What we are storing and how we store might also be subject to change based on the overall implementation down the line.

Similar Classes in the Standard Library:

- std::vector - storage of log entries
- std::accumulate — computing running sums
- std::multiset — potential support for efficient median calculation
- <algorithm> utilities such as std::min and std::max

A list of any other group's C++ classes to coordinate with:

This will most likely be an internal class for the Data that will supply/log the needed data through other classes we write such as OutputManager and ActionLog. It will not be required to interface with C++ classes created by other groups, as of now.

4. OutputManager()

Developed by: Noyonika

Description: A simple logging system for programmers to log events of their choice; often helpful for debugging. It may have multiple levels of output, perhaps "Silent", "Normal", "Verbose", and "Debug" that control whether messages are written to an output file in JSON format, the console, or both. During a live simulation, DataLog will hand off finalized log data and computed statistics to OutputManager, which is then responsible for formatting and persisting this information as a structured JSON file while also emitting human-readable summaries to the console based on the active log level.

Member Variables:

- std::string outputFilePath
  - Current path of the JSON log file to be written.
- LogLevel currentLevel
  - Active log level controlling which messages are emitted.
- std::ofstream outputStream
  - File stream used to write serialized log data and summaries.
- json bufferedLog
  - In-memory JSON object or array that accumulates simulation outputs and insights

Member Functions:

- void SetOutputFile(const std::string& path)
  - Sets the output file path, closing any previously open file.
- Void LogMessage(LogLevel level, const std::string& message)
  - Records a single log message at the given log level, writing to output file and/or console depending on the active log level
- void SetLogLevel(LogLevel level)
  - Sets the active log level, called by World/Agents at the start of a run of the simulation

Error Conditions:

- Attempting to write to an output file before a valid file path has been set.
- Failing to open or write to the configured output file (e.g., permission issues, invalid path).
- Receiving malformed or incomplete JSON/log structures from DataLog or other producers.
- Using an invalid log level

Expected Challenges:

- Designing a flexible log-level system that is simple for other teams to use but still expressive enough for debugging and production runs.

Similar Classes in the Standard Library:

- std::clog
- std::ofstream

A list of any other group's C++ classes to coordinate with:

- Group 12 and 13: Agent base classes (Agents may tag their events with desired log levels or metadata that OutputManager preserves in output), and potentially GlobalManager class.

5. ReplayDriver()

Developed by: Apar

Description: An object that is able to read from the log file of the output manager. Then communicate with the Agent classes to control all actions. The actions that this class will send to the Agents will be a log of a previous dynamically run instance of the experiment.

This driver will only exist when replaying a previously run simulation.

Member Variables:

- A file object that holds a read session from the saved information
- A pointer to the world object

Member Functions:

- Read_Next_Log(string file_name)

    - retrieve the next log from the save file
- Instruct_Agent(Replay_Agent Agent)
    - Use the log that was read to tell a specific agent to take an action. Calls Get_Agent() from world object
- Give_world_info(Global global)
    - Tells the world seed information, the number of agents, and all else startup stuff

Error Conditions:

- If there is no save file present, then we cannot replay anything. Also, if we try to replay a file from a mismatched world version/type, then that will also be an error. For example, trying to load a math world save in an interaction world would result in an error.

Expected Challenges:

- Coordination with Agent classes and World classes

Similar Classes in the Standard Library:

- The closest library is json.hpp that we might end up using. This library is used to read JSON from a file, which is the strategy we are currently attempting to use to store information persistently.

A list of any other group's C++ classes to coordinate with:

- Agent classes and World classes

6. ActionLog()

Developed by: Scott

Description: An object to track the moves made by all active agents in the current World. It will be the live interface between the world, agents, and any other game components. The primary responsibility is for error checking and establishing the interface between producers of log events and the DataLog object. Action Log will be a singleton.

Action Log will only be present during live games (not-replays) and will be owned by DataLog. All agents and the world will have a reference to this.

Member Variables:

- DataLog& datalog

Member Functions:

- void LogEvent(json data)
- LogEvent()
    - checks the log for a valid JSON object and certain key attributes exist such as type, id, time, and log level. LogEvent could also be initialized with a schema and validate all log attempts against it

Error Conditions:

- The LogEvent() function will reject the log attempt if the data fails against the schema ActionLog is initialized with.

Expected Challenges:

- Since this is the main interface for our classes, we will have to coordinate the API for this with other groups for both the game, world, and agents.
- We may want to accept a schema to enforce on certain types.

Similar Classes in the Standard Library:

- **std::clog**

A list of any other group's C++ classes to coordinate with:

- (Internal) Datalog
- (External) Agent, World, Game, Global Manager