EventQueue

Person taking lead - Ho Wang Ho

Class Description

The EventQueue class manages time-based or priority-based events in the game world. It mostly controls in-game events to be scheduled for future execution, enabling cooldowns, delayed effects.

Similar Classes

- Std::priority_queue
- std::chrono

Key Functions

- EventID ScheduleEvent(TimePoint time, Payload data)
- const Event& PeekNextEvent() const
- Event PopNextEvent()
- bool CancelEvent(EventID id)
- bool Empty() const

Error Conditions

- User Error: Cancelling unknown event
- Memory allocation failure

Expected Challenges

- Handling edge cases in event ordering
- Ensuring stable for parallel-time events

FunctionSet

Person taking lead - Henry Yang

Class Description

The FunctionSet class stores a collection of callback functions that all share the same function signature. It is used to register multiple reactions to the same event or trigger,

and when called, it calls each stored function one at a time using the same provided arguments.

Similar Classes

- std::function
- std::vector

Key Functions

- void AddFunction(FunctionType fn)
- bool RemoveFunction(FunctionID id)
- void Clear()
- size_t Size() const
- void CallAll(Args... args) const

Error Conditions

- Programmer Error - Removing a function using an invalid ID or index
- Recoverable Error - A stored function throws an exception during execution

Expected Challenges

- Properly storing different callables while still enforcing one fixed signature

DataMap

Person taking lead - Riley Moorman

Class Description

The DataMap class maps string identifiers to values of arbitrary types. It will be used to allow other classes such as agents, items, and worlds to store and retrieve data. DataMap enforces compile-time type specification and performs runtime type validation to ensure correctness. If a value is retrieved with the wrong type the class will detect a mismatch and will execute an assert. DataMap could be useful for storing any type of data for any type of class. For example Agents could store health, stamina, mana, ect. This will provide a more generic way of retrieving data instead of having to know how classes are structured.

Similar/Related Classes

- std::unordered_map
- std::any
- std::type_index

Key Functions

- template<typename T>
- void Set(const std::string& key, T Value): Used to set a key,value pair in the map
- T& Get(const std::string& key): Used to get a type from the map, if any exists
- bool Has(const std::string& key) const: Used to check if a string pair is in the map
- bool Remove(const std::string& key: Used to remove a string pairing in the map, if any exists. If it doesn't it returns false. If it does and deletes it then return true.
- void Clear(): used to erase all data in the map

Error Conditions

- Missing key: Attempting to retrieve a value that doesn't exist. Programmer error. Handled with assert.
- Type Mismatch: Requested type does not match the stored type. Programmer error. Handled with assert.
- Empty Key Name: Key string is empty (“”). Programmer error. Handled with assert.

Expected Challenges/ Topics to learn

- Implementing safe runtime type checking
- Avoiding unnecessary copies when storing and retrieving data
- Template programming
- Runtime and compile-time type safety

Scheduler

Person taking lead - Caleb Shin

Class Description

Schedules a set of processes based on a “priority” measure and returns the ID of which one should go next. The priority of a process should be proportional to how often it is scheduled. This class can be built to be probabilistic or evenly integrated.

Similar Classes

- std::priority_queue
- std::queue
- std::unordered_map

Key Functions

- void AddProcess(ProcessID id, float priority) - Used to add a process to the scheduler with an associated priority value
- ProcessID Next() - Returns the ID of the next process to be scheduled based on the selected scheduling strategy and current priorities
- bool HasProcess(ProcessID id) const - Used to check whether a process with the given ID exists in the scheduler
- bool RemoveProcess(ProcessID id) - Removes a process from the scheduler. Returns false if the process does not exist, true if it was successfully removed
- void UpdatePriority(ProcessID id, float newPriority) - Updates the priority value of an existing process
- void Clear() - Removes all processes from the scheduler

Error Conditions

- Missing Process ID: Attempting to schedule, update, or remove a process that does not exist. Programmer error. Handled with assert.
- Invalid Priority: Priority value is zero or negative. Programmer error. Handled with assert.
- Empty Scheduler: Attempting to retrieve the next process when no processes exist. Programmer error. Handled with assert.

Expected Challenges

- Designing fair scheduling algorithms
- Implementing weighted or probabilistic selection
- Balancing determinism versus randomness in scheduling
- Managing performance with frequent insertions and removals
- Avoiding starvation of low-priority processes

ExpressionParser - Curtis Lunn

A tool to read in a text string that contains an expression and convert it into a function object that takes a container as input that it uses to draw values from. For example, you

might design it so that it takes a std::map<std::string, double>. If you give it the string “value * scale + offset”, it would generate a function that operates on the map, looking up the keys “value”, “scale”, and “offset”, and then returning the numerical result implied by the expression. You could even have it use a std::vector<double> where an expression might look like the string “{3} + {7}”, where it would produce a function that takes a vector as an argument and adds together the contents of indices 3 and 7, returning the result.