Mohawk Engineering Project

Requirements List

Project #36 - PLC Simulator for Students Training

John Van Loon

Mohawk College

Group #19

Jasper Hanlon

Jesus Perez

Stephen Kim

## Requirements Analysis - FURPS

### Functionality

* Create a C# or C++ application library that exposes a simplified API for interfacing with the Siemens PLC simulator
  + Must be compatible with SIMATIC PLC S7-PLCSIM Advanced V3.0
  + Library must be compatible with student machines running Windows
  + The library must make it possible to build new applications to provide inputs to and consume outputs from the simulator
  + Minimum API Requirements (“Must-have”)
    - Must detect whether PLC simulator is running or stopped
    - Must be able to switch PLC simulator between run and stop mode
    - Must send user input to PLC sim and receive output from PLC sim back to user
    - Must be able to address and interact with a running simulator
      * Can use a constant or predetermined name
  + Optional API features (“Nice-to-have”)
    - Tags
    - Simulating multiple PLC devices
    - Auto-detection and selection of running simulators
  + Additional Features
    - Logging facilities to aid with debugging
* Proof of concept game that uses ~16 inputs and ~16 outputs using our API
  + Game must run concurrently along with the TIA portal and Simulator
    - Game behavior should update to reflect PLC design currently applied to the simulator from TIA portal
  + No specific type of game required. Suggested game examples:
    - Basic simulation/control of a hydraulic cylinder (connect, control, pass digital I/O)
    - Basic simulation/control of a water tank level control (add analog I/O)
    - Perform basic simulation/control of a marble sorter (add simple graphics and interaction)

### Usability

* User must be able to reliably start the game
* Game/API installation must connect easily to PLC sim
* Proof of concept game must be aesthetically pleasing
* API documentation must be accurate and complete for future use

### Reliability

* API must send correct user input and receive correct output from PLC sim
* Game should properly display and work as intended
* Friendly error messages should show in case of crashes
* User logging to show any warnings

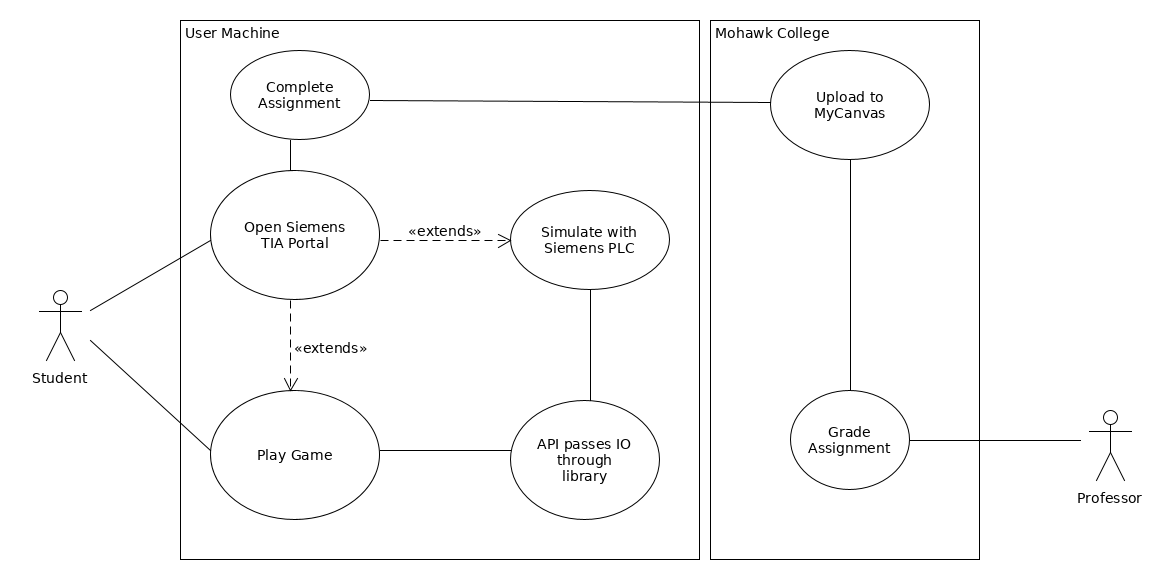
### Performance

* API must pass input/output in real time from PLC simulator
* Game must display API output within a reasonable time
* Game must send API user input quickly to ensure a smooth gaming experience

### Supportability

* High quality documentation for API usage, installation, and development of new games
* Logging capabilities and debug mode
* User activity monitoring

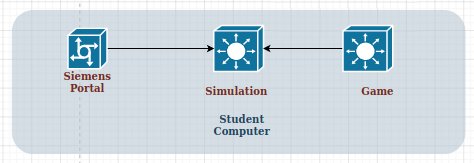
## Use Case Diagrams



## 

## 

## System Architecture/Communication Diagram



## Risk Analysis

**Potential Risks**

1. The Simulator API is buggy or broken for the C++ interface, C# interface, or both
2. The Simulator API is consistently unreliable or unstable
3. The Simulator API cannot be greatly simplified
4. The Simulator API library takes longer than expected to produce, and there is not enough time to develop a game or POC
5. There is difficulty running and using the TIA Portal while the game is running. For example, it may not update adequately quickly, or the simulator must be refreshed manually
6. The game or API crashes randomly
7. Hardware failure, hard drive becomes unusable

**Mitigation Strategy**

1. Use a lower-level interface and avoid using the provided language-specific API libraries directly. Eg. communicate with the simulator directly over TCP-IP
2. Find alternative ways of achieving the same functionality and apply “workarounds” by avoiding code paths or features that invoke simulator instability
3. Shift emphasis on project to documentation. Reduce scope of library, and provide a set of “starter templates” for using the API with a set of minimal functionality
4. Work on the game/POC and the simulator library in parallel. Design the POC based on the planned design of the library before it is complete.
5. Look for workarounds in the API. In the worst case scenario, include instructions in-game on how to use the simulator/portal most effectively while the game is running.
6. Create logging during our development to ensure we know why crashes occur
7. Create regular backups using extra hard drives/VM save states