Candidate Architecture Justification

for

Hydra-TowerDefence

Version 1.1 approved

Prepared by Seenivasan Sashwath Ravilla
Mohamed Shafiq Bin Peer Mohamed
Lim Wi Teow
Heather Chew
Koo Jian Yang
Yeong Wei Xian
Tan Wen Jie
Lim Si Yi

Team Hydra

13/4/2021

Revision History

Name	Date	Reason For Changes	Version
Mohamed Shafiq	8/3/2021	Created document	1.0
Sashwath Ravilla	13/4/2021	Updated project requirements	1.1

1. Chosen Candidate Architecture: Call-And-Return

1.1 Requirements of System:

1. Able to allow for progression through different worlds and levels
2. Able to allow quiz answering and correct scoring of answers
3. Able to pass control from one component (gameplay subsystem) to another component (quiz subsystem)
    a. This is for ease of development and to avoid creating a god-like component that controls all other components
4. Must allow for concurrent execution
    a. Multiple players must be able to play simultaneously, against each other in a PVP mode
5. Scalable to allow for increase in question bank
6. Needs to be interactive to keep gameplay experience smooth and responsive

1.2 Common Architecture Styles

1. Independent Components
2. Data Flow
    a. Batch Sequential
    b. Pipe-And-Filter
3. Call-And-Return
    a. Main Program and subroutine with shared data
    b. Layered system

1.3 Justification:

Needing our system to give “correct” results every time, with very little room for error (since we had a quiz and grading system), we had to rule out using the Independent Component system, since it is known that achieving reliable correctness is difficult on this particular architecture.

To make development of the game easier, we sought to divide the workload evenly and keep components independent, with all components only being common in their ability to pass control to another component if necessary. This helped us avoid needing to create a god-component/super component that had to handle control over the entire application. Having such a component would have slowed down development by needing constant modifications as other components were updated, and also helped reduce troubleshooting difficulties. This meant that the Client-and-Server & event based implicit invocation system were both ill-suited for our use.

Since we needed our system to be responsive (point 6), we ruled out using the Pipe-and-Filter architecture design as well, as it is known to be poor at handling interactive applications. Since we were developing this game as a large group of 8 members, we believed that modularity was important and would help speed up the development lifecycle of each independent component. As a result, by process of elimination and our specific development needs, we settled on using the Call-And-Return architecture, focusing on the main-program-and-subroutine with shared data substyle.