STOP TALKING AND EVERYBODY EXPLODES EXTENDED

Quinton Randall

Grand Valley State University

Goal:

Compare and contrast the differences in development. These development types are development-less (bootlegging), and following an Agile Layered Architecture approach. This will be based on the production of a text-based bomb defusal game based on the game “Keep Talking and Nobody Explodes”(KTANE), by the 2nd of December.

Definition:

I am comparing the software development process of a text-based bomb defusal game based on the game “Keep Talking and Nobody Explodes” using specific software development methods and architecture, and freehanding/ bootlegging(no specific method).

Functional Requirements:

- The program shall provide the player a description of the current bomb and associated modules.
- The program shall prompt the player to choose one of 3 random modules at the start.
- The program shall prompt the player to input another guess if their input is correct.
- The program shall issue the player a strike if their input is incorrect
- The system shall terminate after the player accumulates 3 strikes.
- The system shall display a failure message if the player fails, or gets a module incorrect.
- The system shall display a success message if the player successfully completes a module.
- The wire module shall correctly interpret wire colors, handle different wire configurations, and respond appropriately to user inputs.

- The morse code module shall correctly interpret the generated morse code, and validate them against user input, and expected outputs.
- The button shall correctly interpret button color, label, and respond appropriately to user inputs
- The colored strip module shall correctly interpret strip color, and respond appropriately to user inputs.

Non-Functional Requirements:

- The game shall provide clear prompts and error messages to guide the player through interactions with modules.
- The game shall maintain accurate tracking of strikes across all modules.
- Response time for inputs and module transitions shall not exceed 1 second.
- The game shall be structured to allow for easy updates to add new modules or adjusting existing ones without disrupting core functionality.
- The system shall be designed to accommodate additional modules and complexity for future versions.

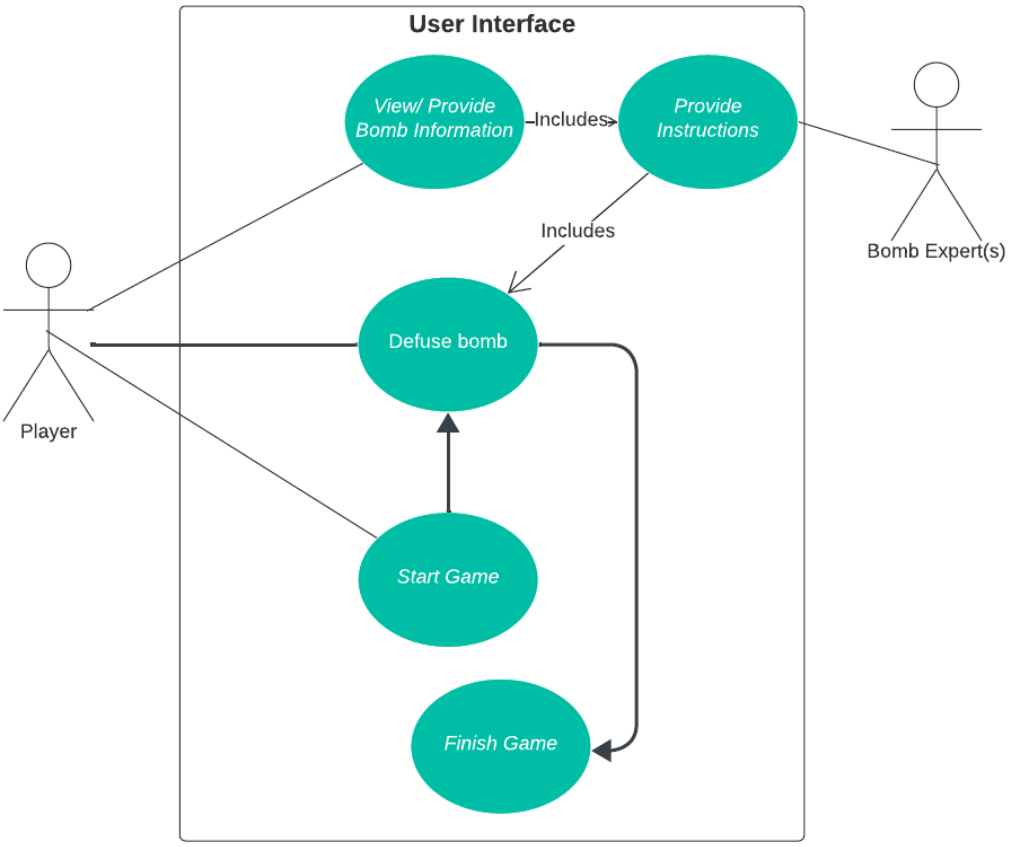
Previous Design/Implementation:

System	**System Context**	**Irrelevant Environment**
User Interface	Player	Room Temperature
Game Server	Bomb Experts	Homework
Database	Controller	Snack Jar
	Voice Chat(communication)	

Use Cases	**Actors**
Start Game	Player (Seed bomb)
View/provide Instructions	Bomb Expert (Sees instructions)
Defuse Bomb	
View/provide bomb information	

STAEE4

UCD:



Use case: Start Game

Actors: Player

Description: The player initiates a new game.

Use case: View/ Provide bomb information.

Actors: Player

Description: The player views the bomb module and communicates its contents to the bomb defusal expert.

Use case: Provide instructions.

Actors: Bomb Expert

Description: The Bomb Expert provides instructions based off of the player's information about the bomb/modules.

Use case: Defuse Bomb

Actors: Player, Bomb expert

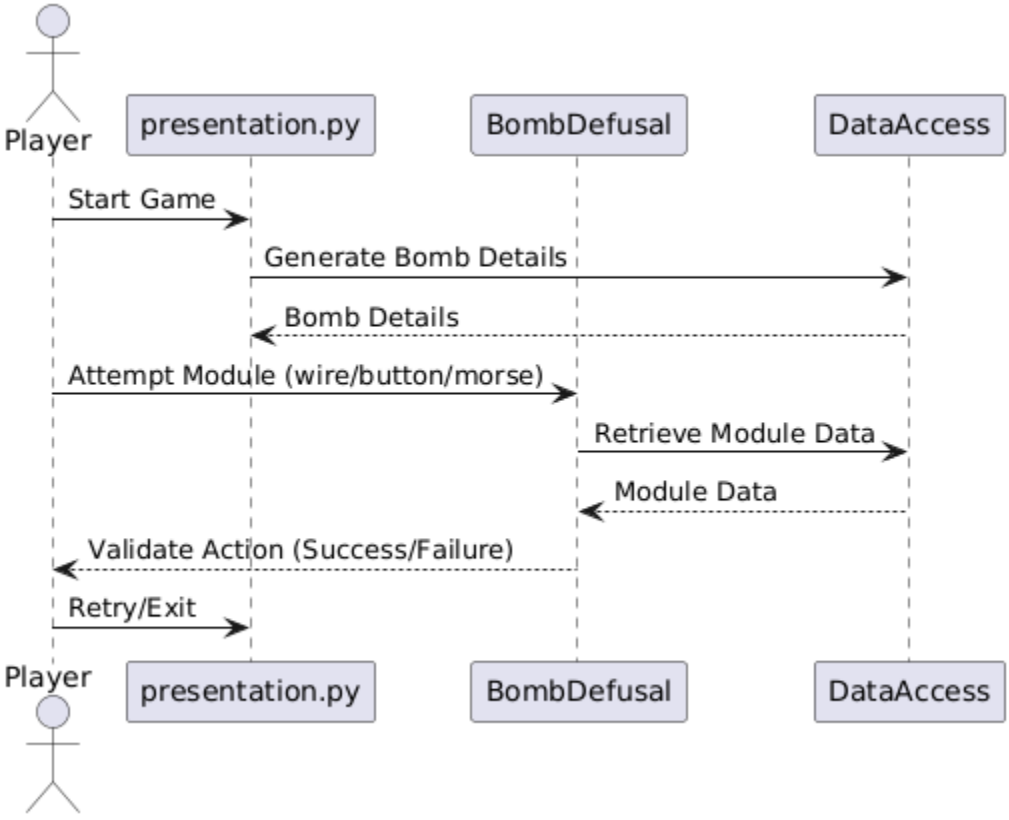
Description: The bomb expert relays steps to the player based on the information provided to defuse the bomb.

Use case: End Game

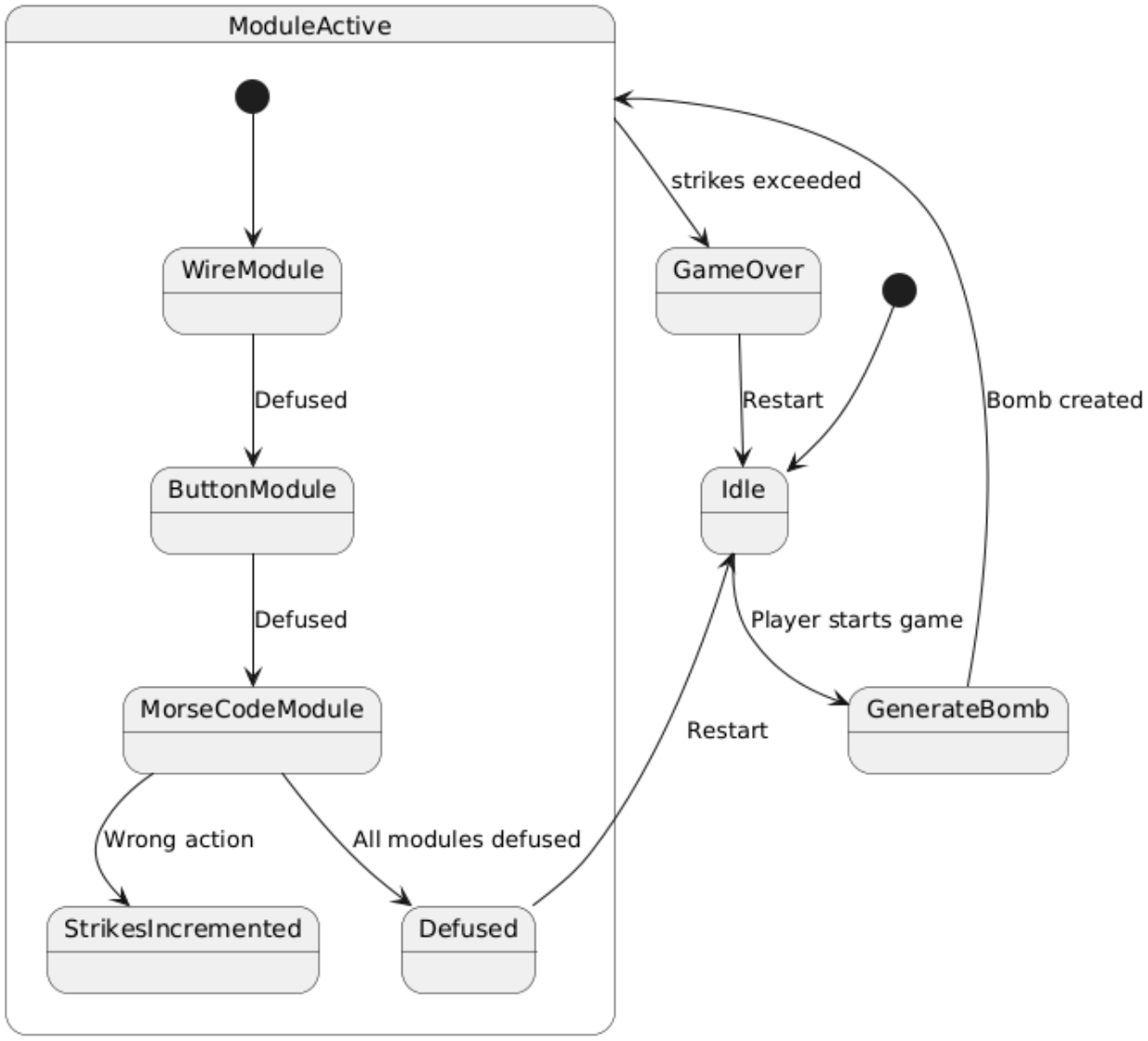
Actors: Player

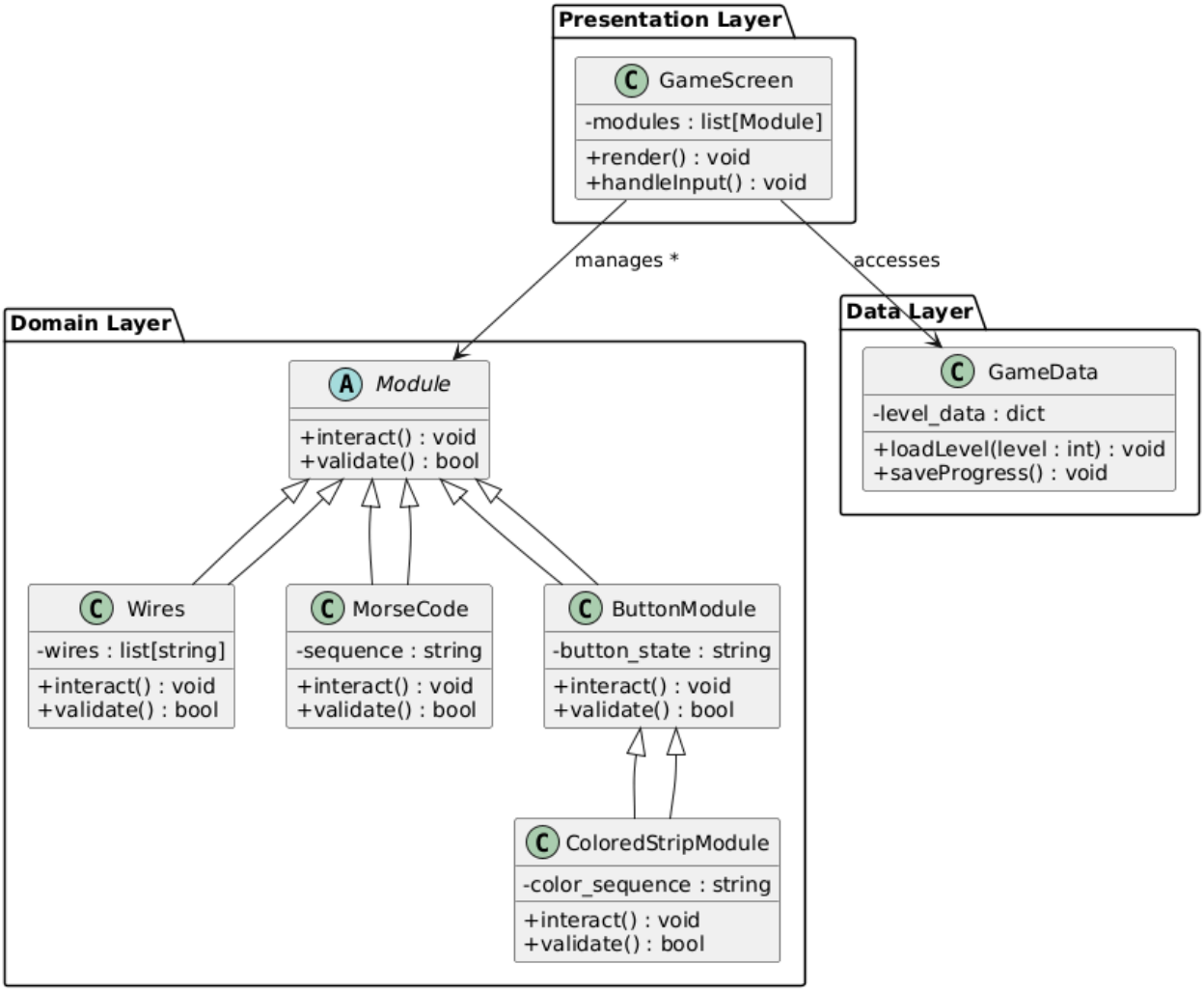
Description: The player successfully defuses or triggers the bomb, concluding the game.

Current Design/Implementation:



STAEE6


ModuleActive
WireModule
Defused
ButtonModule
Defused
MorseCodeModule
Wrong action
All modules defused
StrikesIncremented
Defused
strikes exceeded
GameOver
Restart
Idle
Player starts game
GenerateBomb
Bomb created
Restart



Above are my class, state machine, and sequence diagrams. Building these high level diagrams, including functional and non-functional requirements helped streamline the process in development of this system. This greatly increased the productivity, and cohesiveness of the project. I chose the layered architecture style for its modularity, and to try something new. The layered architecture design pattern is a software design pattern that organizes applications into separate layers that can be managed and maintained independently. This allows for enhanced scalability, and testability by isolating functionality into defined levels, each with their own set of responsibilities. I moved away from the use case diagram as I wanted to see how new methods would affect or enhance the process, and I did not want to utilize any feature in the previous

iteration, keeping a consistent control. The sequence diagram and the requirements were the most helpful in developing the new version of the game. It made it easier to follow a step by step process. Reverse engineering in this type of way was unique, and shows the functionality and modularity of a layered architecture approach.

With the new architecture It reduced many of the problems that the previous iteration had. There was no problem in coloring wires, or indicators. There are now less redundant modules(duplicated code), and with the implementation of code smells, everything is more readable. Initially I had everything be produced in its own variable, and initiated it in many places, when it was only ever needed for the specific module, so there was no need to initiate it in one portion, and then again in the function where it is needed. This removed unnecessary complexities, which in turn reduced the length of lots of code. One of the new problems introduced was incorrectly categorizing each piece of information into the correct file to prevent circular imports. A circular import happens when two or more files are retrieving information from each other making them both dependent on one another. This problem occurred initially in the wire module as I was incorrectly categorizing the view and the actual interactions of the wire module. I would take the generated wires from the data, display it, initiate the logistics, but go back to display for more information, and try to finish the logic. This was easy to fix however, especially with this type of architecture, as it was intuitive with the layout. Along with the design smells, I also kept in mind different code smells. Going through my code, I was constantly evaluating what can be improved, such as comments. I am a firm believer in the “You should not need comments in your code if your code is good”. This is likely not the exact words, however it fits my purpose. Just by reading the code, you should be able to know exactly what it does. I feel

like I implemented modules well enough to know how everything works, the logic, and the verbiage connected to them.

The biggest differences between the approaches were the ease of use, and the flow of design. Initially, to get information and build the game, it was more so reverse engineering the actual game. This took a lot of playing the actual game, watching videos of others, and data mining. This took a significant amount of time as my only real requirement was to recreate what I could find from the gathered information. With functional and non-functional requirements, it was a lot easier to know what information or what techniques should be implemented. I was able to categorize all aspects into different levels. The sequence diagram let me know how, when, and where I should implement the modules. As far as the process goes, it follows, “what is the functional requirement”, “what dependencies are needed(data)”, “what should it look like(presentation)”, and “how should it react(business)”. This streamlined process made implementing the entire system, and adding more features much easier.

Resources:

Rules in PDF Of implemented game:

https://www.bombmanual.com/print/KeepTalkingAndNobodyExplodes-BombDefusalManual-v1.pdf

MISHRA, A.; RAY, A. K. A Novel Layered Architecture and Modular Design Framework for Next-gen Cyber Physical System. ***2022 International Conference on Computer Communication and Informatics (ICCCI), Computer Communication and Informatics (ICCCI), 2022 International Conference on****, [s. l.], p. 1–8, 2022. DOI 10.1109/ICCCI54379.2022.9740757. Disponível em: https://research.ebsco.com/linkprocessor/plink?id=eb7ed8a7-a3d0-3667-bacb-ae16cc9384bf. Acesso em: 3 dez. 2024.*

Pereira, J. F. C., & Cruz, A. M. R. D. (2023). Approach for Fast Growing Software Systems using layered architecture : Scaling an architecture with minimal refactoring. 2023 18th Iberian Conference on Information Systems and Technologies (CISTI), Information Systems and Technologies (CISTI), 2023 18th Iberian Conference On, 1–6. https://doi.org/10.23919/CISTI58278.2023.10211943