UPDATED CSC207 DESIGN DOCUMENT

# Rhythm Labyrinth

*Version 0.1 – 2023.11.03*
*Created 2023.11.03*

Members of Group 6

| | | | |
|---|---|---|---|
| Ashfaq, Raza | Mohammed, Ayman | Wang, Samuel | Yang, Kairui |

## GitLab Repository:

https://mcsscm.utm.utoronto.ca/csc207_20239/group_6

## SECTION 1: PROJECT IDENTIFICATION

Our group (Group 6) project is best described as an extension of this semester's AdventureGame game software on multiple fronts. While every member had a unique perspective towards implementation, through the creation and review of numerous User Stories, the team had ultimately settled on four Design Patterns that best-realized the functional potential of implementation motivations.

Firstly, our team wished to incorporate both cognitively-demanding Puzzle games as well as reflexively-challenging Battle games to enhance the player experience. The Puzzle game tests the player's ability to memorize a visual sequence of squares on a grid that increases in length each round, while the Battle game challenges the player's ability to click on randomly-appearing circle widgets within a decreasing span of time. The implementation choices behind these mini-games lend themselves to the Singleton and Command Design Pattern, as will be discussed in further sections. Secondly, our team wished to implement a game that gave the player a sense of influence over their outcome. This project motivation, as will be seen under further scrutiny, was operationalized via the Factory and State Design Pattern. These patterns will then be leveraged to assess the Player's performance on aforementioned mini-games and ultimately track their progression down unique developmental "routes".

Finally, our team has constructed two accessibility User Stories in regards to implementing our project in consideration for People with Low Vision (further described in the User Stories). We expect to incorporate them into our GUI designs in further sprint iterations. Overall, what resulted is an inclusive and well-rounded extension of the AdventureGame software this semester.

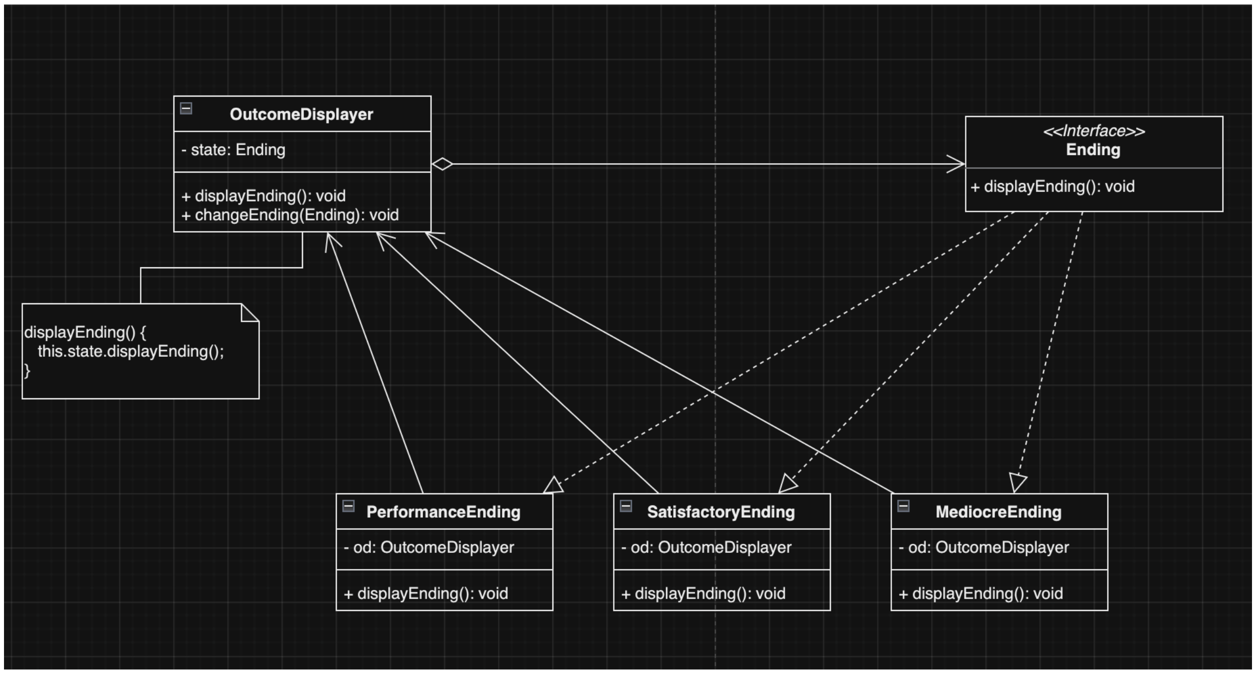
## SECTION 2: UPDATED USER STORIES (BASED ON WHAT WAS ACTUALLY IMPLEMENTED)

| Name | ID | Owner | Description | Acceptance Criteria | Implementation Details | Priority | Effort |
|---|---|---|---|---|---|---|---|
| Existence of Different Routes | 1.1 | S. Wang | As a video game player who is enthusiastic about multiple endings, I want a continuous stream of in-game events to have a gradual, incremental impact on my adventure outcome so that I can actively shape my adventure outcome. | Given that I am a video game player who is enthusiastic about multiple endings, when I perform within a certain bound on a mini-game, then I progress down a certain, respective developmental route. | There will exist different classes for different developmental routes. Likely another set of classes will exist for each type of game (Puzzle or Battle) to interpret their resulting scores to determine what specific developmental route that the player has contributed to for that round. | High Priority - influences progress of the game | 2 |
| Minigames | 1.2 | R. Ashfaq | As a person who enjoys playing games with engaging gameplay, I want there to be minigames to play, specifically puzzles and battles, so I can have engaging interactions while playing. | Given that I am a person who enjoys playing games with engaging gameplay, when I play the game, I will encounter minigames such as a battle or a puzzle. | We will create two types of minigames, a puzzle and a battle, that a player can engage in | High | 3 |
| Display GUI | 1.3 | Kairui Yang | As a person who enjoys playing games with an immersive interface, I want the game to have a GUI that displays everything I can interact with and shows the entire gameplay, so I do not have to see the backend workings of the game and ruin my immersion. | Given that I am a person who enjoys playing games with an immersive interface, when I launch the game, I should be able to play the entire game exclusively on a frontend interface. | We will have a GUI display for each required scene, including one for each of the minigames, a menu panel, and an ending screen. | High | 3 |
| Tutorials | 1.4 | A. Mohammed | As a novice adventure game player, I want to be able to engage with the controls and mechanics of the game in a low-stakes environment so that I can familiarize myself with the game with ease. | Given that I am a player that is unfamiliar with similar games, when I start the game for the first time, I should start in a tutorial room that teaches me the controls and mechanics of the game. | When the game starts, the first room should act as a tutorial room - a room that shows the controls of the game, as well as the primary mechanics of the game. The player might be prompted with a “is this your first time playing?” screen that will allow the player to skip the tutorial if they have already played through it once before. | Low priority | 2 |
| Colour Changing | 1.5 | Kairui | As a person who has low vision accessibility needs and likes video games, I want the ability to change the colour of the HUDs and/or text by inverting the colour of the scene so that it may be easier to see the game. | Given that I am a person who has low vision accessibility needs and likes video games, when I open the menu, then I can invert the button by pressing an invert colour button. | We will have a button in a menu panel that we can press and it will invert the colour of the scene by using a blend. | High | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Clear Puzzle Goals | 1.6 | K. Yang | “As a beginner player with no experience of the game, I want the goals of the puzzles and the correct sequence to be clearly outlined so I know what I must do to complete the puzzle without getting frustrated trying to figure out how to even start.” | “Given that I am a beginner player with no experience of the game, when a puzzle sequence begins, I am told what task I must accomplish and what the end state of the puzzle should be.” | A simple text will display “Input the sequence of buttons shown to you in the correct order”.<br>The buttons light up one after another with distinguishing color to indicate the order of the sequence. | High | 3 |
| Variation in gameplay | 1.7 | Ayman | “As a player of the game who enjoys puzzles and experiencing variation in games, I want each room to either include a battle with an enemy, or a puzzle so I can have variation when I’m playing and be more engaged.” | “Given that I am a player of the game who enjoys puzzles and experiencing variation in games, when I enter a room, I either get a battle I must beat to move on, or a puzzle to complete.” | Each room will have a predetermined main challenge they must beat. This challenge will either be a battle or a puzzle. | Medium | 2 |
| Performance Based Route development | 1.8 | S. Wang | “As a player of the game who enjoys performance based outcomes, I want the extent of my success with the puzzles or battles to determine the magnitude of progression down my adventure route so that I can actively shape the fate of my adventure.” | “Given that I am a player of the game who enjoys performance based outcomes, when I complete a puzzle or battle, depending on how well I completed the puzzle, the magnitude to which I progress down a certain route corresponds to the quality of my completion.” | The classes in 1.1 that interpret the scores from Puzzle/Battle minigames will also track the magnitude that the player has progressed down a specific developmental route. There will exist a unique class that keeps track of the player’s progressions down all developmental routes at all times during the adventure. The route that has been most prominently progressed will be the one chosen for the final outcome when the game finishes. | Medium | 3 |
| Achievable end goal | 1.9 | S. Wang | “As a player of the game who enjoys more conventional and less restricting playing styles, I want to only have to complete a minimum number of puzzles and/or battles in any combination that I choose so that I can complete the game in as freely a manner as possible.” | “Given that I am a player of the game who enjoys more conventional and less restricting playing styles, when I wish to move on to the final outcome of the game, the game verifies if I have met the minimum completion requirements.” | Players will be checked for the number of puzzles/battles that they have completed upon attempting to move onto the final outcome of the game. | High | 2 |
| Difficulty scaling | 1.10 | A. Mohammed | “As a player of the game who enjoys more difficult challenges as I improve, I want the difficulty of the puzzles and battles to increase as I play for longer so that I can enjoy a cognitively challenging game.” | “Given that I am a player of the game who enjoys more difficult challenges as I improve, when I play more puzzles or battles, the iterations become increasingly long and complex for the puzzles and the battles require quicker reaction times.” | There will be nine buttons placed on a 3 x 3 grid, in which a sequence of buttons will flash, one by one. Each sequential round will consist of the same sequence of button flashes plus a new flash at a randomized button location. Each button will be mapped to a specific number in a static Map, and the sequence generated by the user input will be verified for a match with the randomized button sequence.<br>For the battles, the circles | Low | 2 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | that appear will disappear quicker, demanding faster reaction time from the player as they play for longer | | |
| Defined Battles | 1.11 | Kairui/Raza | "As a player of the game who enjoys playing battles, I want the battle to test my reaction speed and motor skills (i.e. ability to click on targets within a limited time frame) so that I can enjoy the thrill of the moment." | "Given that I am a player of the game who enjoys battles, when I want to enjoy a game that invites thrill and quick focus, then I engage in quick rounds of the battle minigame." | Within a gridPane, (or some widget), players will need to click on a circle button that will appear at a randomized location on this widget before a timer runs out. If they do this successfully, then they successfully "deal damage" to the enemy. If they fail, then their "hp bar" reduces in points. As the rounds proceed, the timer countdown reduces in capacity and the damage dealt/lost increases. | Medium | 3 |
| Possible Moves | 1.12 | Kairui | "As a player of the game without a consistently functioning keyboard, I want the possible moves within each room to be displayed as buttons to press instead of typing the moves so I can still play the game with ease with just a mouse" | "Given that I am a player of the game without a consistently functioning keyboard, when I'm in a room, the possible moves for me to make (Up, Down, North, etc.) are displayed for me to see and I can click one of the moves to perform that action." | We will still use something similar to "getMoves" from the current room, except we will remove the input text field and replace it with a series of buttons positioned in their respective directions, which displays all of the possible movesets within the room. | High | 2 |
| Gameplay | 1.13 | Ayman | As a player who enjoys games with a sense of cohesion, I want to have a sense of story/progression when I play the game so I can feel engaged with the game. | Given that I am a player who enjoys games with a sense of cohesion, when I play and as I go through the game, I want to experience a full cohesive story driven experience. | We will redesign the original implementation to accommodate for the new version of the game and we will create a new storyline for it. | High | 3 |
| Defined Puzzles | 1.14 | Raza | As a player who enjoys puzzles in games, I want the puzzles to test my memory through memorizing a sequence of patterns so I can feel challenged and engaged with the game. | Given that I am a player who enjoys puzzles in games, when I play a puzzle, a 3x3 grid of buttons will appear and give me a sequence of buttons to memorize. | We will make a 3x3 grid that generates a random sequence of buttons and asks the user for the same input, evaluating whether the user input is correct or not. | High | 3 |

## *Design Pattern #1: State Pattern*



Overview: This pattern is used to implement the creation of the *OutcomeDisplayer*.

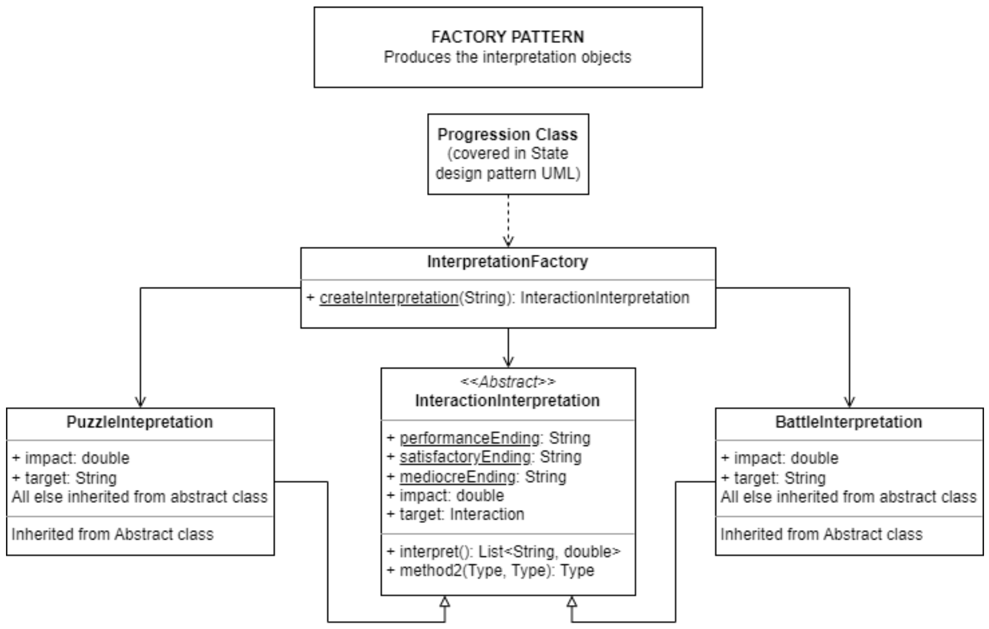
This design pattern was chosen to incorporate solutions that address User Stories 1.3 and 3.11, which describe a request for the player to be able to gradually and incrementally shape the outcome of their game based on their performance on puzzles and/or battles throughout the game. This request was realized through the *OutcomeDisplayer* class.

As further UML Diagram overviews will discuss, the game's mechanisms will determine (via the player's performance in puzzles/battles) how far they progress down each of three routes. Based on the most prominently progressed route, the *OutcomeDisplayer* class executes its *changeEnding(Ending)* method to select the appropriate outcome tailored to the player's route. When the player has completed the game and is ready to move onto the final ending scene, *OutcomeDisplayer*'s *displayEnding()* method is called, which actually calls *this.state.displayEnding()*. To further elaborate, the *state* attribute is actually the Ending selected by the *changeEnding(Ending)* method, which means that *OutcomeDisplayer displayEnding()* actually calls the *displayEnding()* method of its attribute *state*.

These implementation choices, characteristic of the State Design Pattern, promote the reuse of the same *displayEnding()* method from *OutcomeDisplayer*, reduce redundant copying of code and conditionals for different states (i.e. game outcomes), and promote the use of class abstraction, reducing coupling between concrete classes. Hence, the State Design Pattern for *OutcomeDisplayer* is a simple and practical solution for the User Stories that call for a game with multiple endings - or states - determined by player performance.

## *Design Pattern #2: Factory Pattern*

Overview: This pattern is used to implement the creation of an *InteractionInterpretation*.


FACTORY PATTERN
Produces the interpretation objects
Progression Class
(covered in State design pattern UML)
InterpretationFactory
+ createInterpretation(String): InteractionInterpretation
<<Abstract>>
InteractionInterpretation
+ performanceEnding: String
+ satisfactoryEnding: String
+ mediocreEnding: String
+ impact: double
+ target: Interaction
+ interpret(): List<String, double>
+ method2(Type, Type): Type
PuzzleIntepretation
+ impact: double
+ target: String
All else inherited from abstract class
Inherited from Abstract class
BattleInterpretation
+ impact: double
+ target: String
All else inherited from abstract class
Inherited from Abstract class

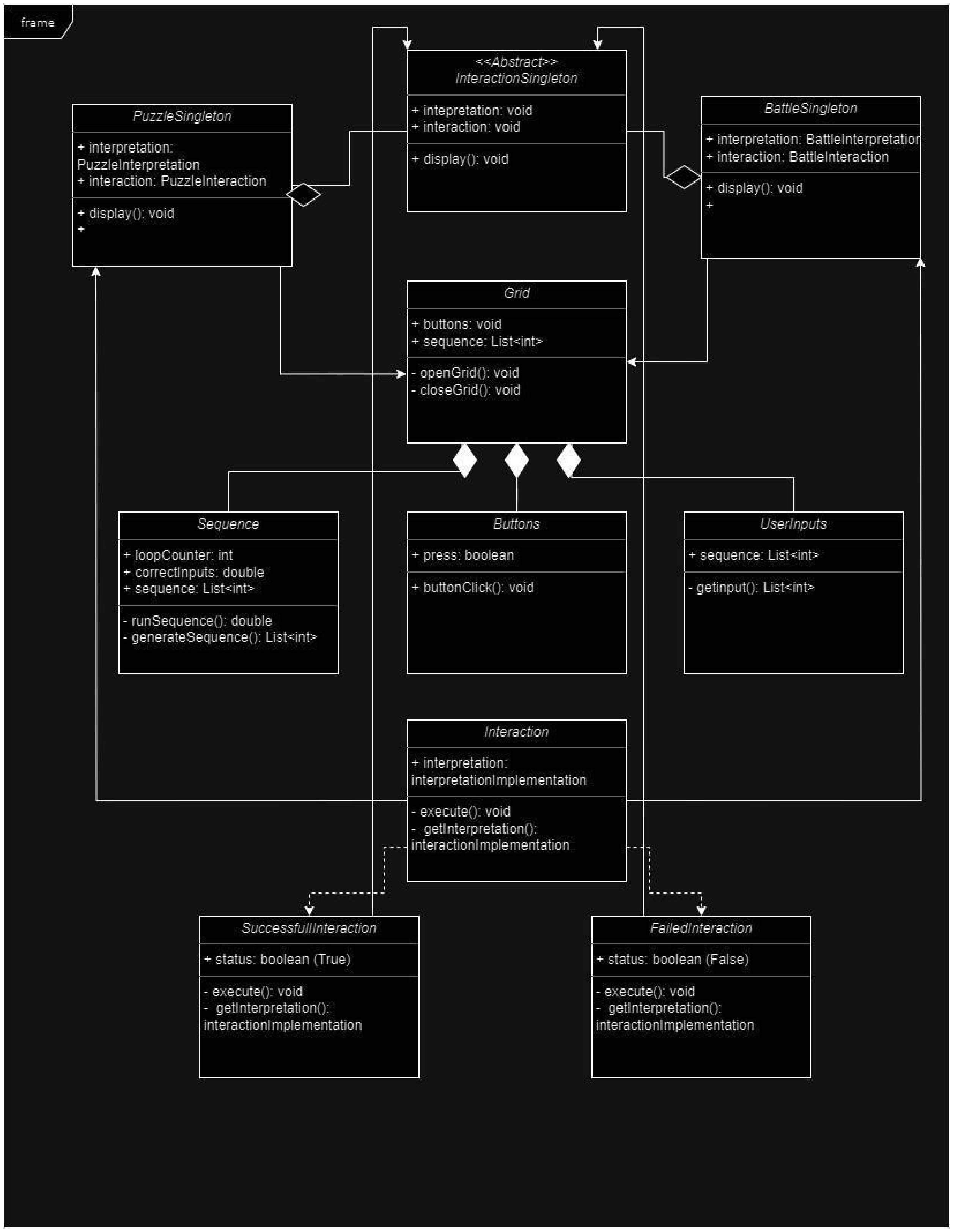
**Implementation Details:**

The main focus of this diagram is on the *InteractionInterpretation*. For our game, every battle and puzzle will have its own *PuzzleInterpretation* or *BattleInterpretation* to perform the analysis on the player's performance in each puzzle and battle and update the outcome at the end. Since each Puzzle and Battle will create a new *InteractionInterpretation* every time, we thought it would be more efficient to create a factory class for creating the *InteractionInterpretation.* This way, if there are more *Interactions* that get implemented as the project scales up, we can continuously rely on the factory to make the appropriate *interpretations.*

How it works is, every time a *PuzzleInteraction* or *BattleInteraction* is created, the constructor will call the static *createInterpretation()* method in the *IntepretationFactory* class, passing in a string to specify the creation of the appropriate *InteractionInterpretation*.

This part of the overall project addresses the user stories for the multiple endings, as well as the user stories for performance-based outcomes.

## *Design Pattern #3: Singleton Pattern  (not implemented)*

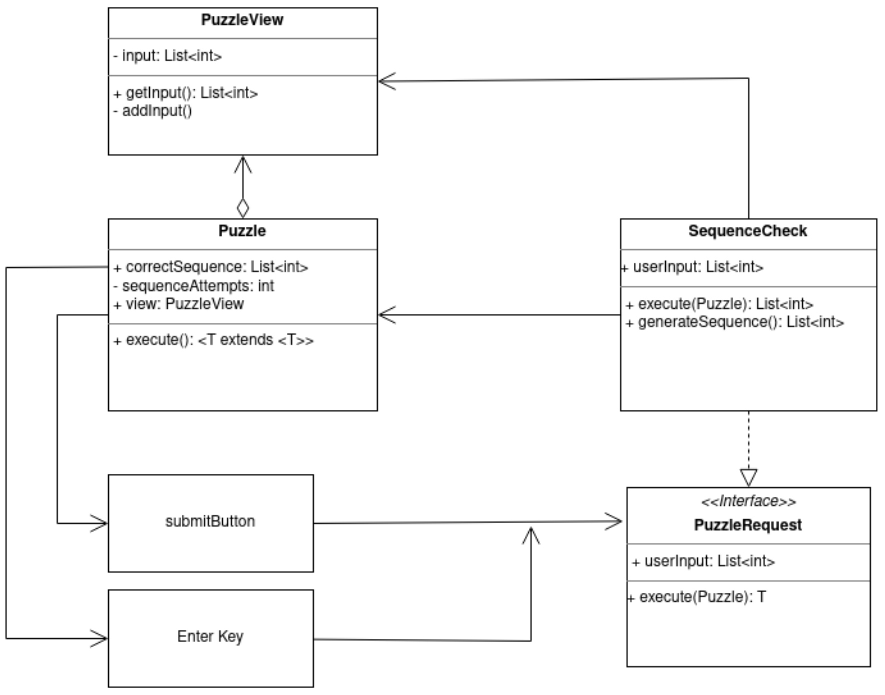


ImplementationDetails: This UML design focuses on the usage of the grids and interactions for both the puzzles and the battles. Using the abstract class InteractionSingleton, it is composed of the two PuzzleSingleton and BattleSingleton which separates the two based on their properties.  It shows some of the subclasses and attributes of each puzzle and battle class such as the sequence, user inputs, and the buttons that will be used as the interactable part of the adventure game. All of these attributes and subclasses will be operating on the grid class, which is the playable space for the player to interact on for the puzzles and battles.

The main focus of this Singleton pattern is that it all loops back. The player goes through the puzzle or battle and does their attempt. If the user attempt matches up with the given sequence, then this is declared as a successful interaction and will be declared as a SuccessfulInteraction. Otherwise, if the user attempt and the sequence are not connected, then the interaction will fail and will be declared as a FailedInteraction. After the interaction, whether successful or not, it loops back to the start of the InteractionSingleton class as the process continues for the remaining puzzles and battles. This process continues until the player has reached a specified ending for the game.

We chose this because it saves costs and resources so that the program does not have to constantly create new grids and playable spaces for puzzles and battles. Instead reusing the grids creates a standard for what the puzzles and battles can refer to and gives the game a sense of familiarity for the play space.  It gives the game consistent gameplay which allows the player to be able to practice as they go through the game in varying difficulty of the interactions. Thus this design pattern can ensure that only one instance of the battle and puzzle classes exists to help with efficiency.

## *Design Pattern #4: Command Pattern (not the same)*



**Implementations Details:**
This UML describes the functionalities of the model for the Puzzles, and more specifically, the interaction of classes in order to communicate the user input to an external command, (classes that extend the **PuzzleRequest** interface), here a SequenceCheck, to check if the user-inputted sequence matches the sequence previously displayed, it generates a new sequence as required, or send feedback otherwise.

The UML Design uses the Commands design pattern in order to focus on the requests to check for the correct pattern inputted into the puzzle into a stand-alone object (SequenceObject) that contains information about the request, and execute logic to then pass information based on the kind of request it undertakes, streamlining the design and allowing for easy extensibility for future complex puzzle mechanics.

Clicking the button or the Enter key instantiates a SequenceCheck object that takes the instance of the Puzzle class and stores the necessary details required for the check. It then executes, and checks if the input equals the correctSequence.

If it does, it generates a new sequence if the puzzle has another sequence for the player to remember, returning that value to the Puzzle class for it to display to PuzzleView, and finally communicates the status of the execution. If the value is incorrect, it adds one count to the loopCounter attribute, and it returns an error status to the Puzzle instance, indicating that the Player must attempt the sequence again.