Detective Quest

Version One – 2023.11.06
Version Two – 2023.12.05
Created 2023.11.04

Project Name

Sehajrattan Dhillon
Christian Kevin Sidharta
Wahib Ali
Wage Mareto Ghazanfar

GitLab Repository:

https://mcsscm.utm.utoronto.ca/csc207_20239/group_69

Simplified Project Changes:

- Changed from doing five “buildings” to one “building” (The mansion and this is where the murder happens)
- Changed from suspects to be the servants/maids and family of the mayor

SECTION 1: PROJECT IDENTIFICATION

The motivation behind the project, Detective Quest, is to create an engaging and immersive adventure game that offers the player a thrilling experience as they step into the shoes of a detective tasked with solving a murder mystery. This project seeks to tap into the human fascination with mystery and intrigue, providing players with an opportunity to immerse themselves in a captivating narrative that challenges their problem-solving skills, deductive reasoning, and attention to detail.

Detective Quest enhances the functionality of existing adventure games by combining the elements of a detective thriller with strategic gameplay. While adventure games have traditionally featured exploration and puzzle-solving, Detective Quest takes these elements to a new level by introducing a time-bound, story-driven experience. Key additional enhancements include, time constraints, character interactions, and deductive gameplay. Altogether, Detective Quest is motivated by the desire to provide an engaging and captivating game experience that satisfies the players' curiosity for solving mysteries by implementing additional features not commonly found in adventure games.

SECTION 2: USER STORIES

Name	ID	Owner	Description	Implementation Details	Priority	Effort
VisualizingRoom	1.1	Sehaj	As a user of a Detective Quest Game, I want to be able to visually see the rooms when I travel between them.	Create files for the new rooms that are related to the game.	1	2
VisualizingObject	1.2	Christian	As a user of a Detective Quest Game, I want to be able to see the Object in multiple Rooms, so that I can pick the Object in the Room	Create a new files for new objects that are related to the game	1	4

RoomConnection	1.3	Wahib	As a player, I want navigate to across the mansion in a way that makes logical sense, so that I can uncover more clues.	Remade the entire rooms.txt, file, with different room names, descriptions, etc. Also replaces room images to match the rooms.	1	2
Timer	1.4	Reto	As a user, I want to know how much time is left during a time-constrained game, so I can gauge my progress and make informed decisions.	Create a time class that keeps track of how much time is left. Then make a time object and a Timeline object inside AdventureGameView to update the time object every second. Then make a Label to inform the user how much time is left for them.	1	4
CluesFeature	2.1	Sehaj	As a user of a Detective Quest Game, I want to be able to interact with clues dropped throughout the game in the same manner I do with objects.	Create a new AdventureClue interface similar to AdventureObject, and update clues in both the inventory and objects/clues in room as the user makes their way through the game by getting updates from an observer.	2	3
TimerEnds	2.2	Reto	As a user, I want to see what happens when the timer ends so I can progress through the game and finish it.	Show the end game screen when the timer ends by using the method Christian created in his End_Of_Game implementation.	1	2
End_Of_Game	2.3	Christian	As a user of a Detective Quest Game, I want to be able to choose the Killer at the end of the Game, so I can know what is the ending of the Game based on my decision	Create an End_Screen_View class with suspectList and endingText attributes where it will display the right text based on the users' choice of killer in the game	2	4

SuspectFeature	2.4	Wahib	As a user in a detective game, I want to have potential suspects that I can	Create a suspects.txt and updated AdventureLoader.java and AdventureGame.java to handle the parsing of the text file.	2	4
ColourInverted	3.1	Sehaj	As a user of a Detective Quest Game and a person with a visual impairment, I want to be able to view the game in contract mode where everything is in a lighter tone.	Create an End_Screen_View class with suspectList and endingText attributes where it will display the right text based on the users' choice of killer in the game	3	5
Sound	3.2	Reto	As a user with a visual impairment, I want to be able to visualize images by hearing the room description articulated.	Use the Free Text-to-Speech library to read the room description. Implement this by replacing the current ArticulateRoomDescription inside AdventureGameView.	3	4
Map	3.3	Christian	As a user, I want to know where I'm currently at in the game, so that I can navigate easily through the rooms.	Make an image of the location for each room and show it up in the GUI by adding an ImageView to gridPane.	3	2

Contrast Ratio	3.4	Wahib	As a visually impaired player, I want to have clear contrasting text of at least a 4:5:1 ratio, so that I can read in-game text clearly.	Ensure that all labels, images and backgrounds maintain 4:5:1 ratio upon instantiation and updates.	3	4

Some examples of **acceptance criteria** for these user stories are below.

Name	ID	Description
Sound	1.1	Given I am a non-sighted user When I enter a room in the game, Then I hear an audio description of the room, including its layout, key objects, and any relevant clues.
Visitor	1.2	Given the detective is in the game, When the detective attempts to navigate between rooms, Then the game should allow the detective to move to different locations using appropriate commands or in-game actions if the detective has the appropriate clues, ensuring seamless room transitions. Given the detective is in a room, When the detective explores the room, Then there should be interactive objects or clues that the detective can examine or interact with as part of their investigation, adding depth and engagement to the game. Given the detective interacts with objects or clues in a room, When actions are performed within a room, such as examining objects or gathering clues, Then these actions should have a tangible impact on the detective's progress or lead to new discoveries, ensuring meaningful gameplay.
Inventory Update	1.3	Given that I am a player in the game, When I pick up an object, then my expectation is that the inventory updates immediately to include the newly acquired item. Given that I am a player in the game, When I drop an object, then my expectation is that the inventory updates promptly to reflect the removal of the dropped item. Given that I am a player in the game, when I access my inventory, then my expectation is to see a clear and organized list of all collected objects, including their names and relevant details.
Command Dependency	1.4	Given there’s a command I want to add, I make a command class that uses the command interface, I just need to change what’s necessary in the command interface.

Undo	2.1	Given I misclicked a command, When click the undo button, The game state reverts to before I misclicked.
Available Commands	2.2	Given I want to know what commands I can do, I open the game instruction, I now know there's a command to get available commands, I execute that command, I see what commands I can do.
Command Description	2.3	Given I want to know what this command does, I open the game instruction, I now know there's a command to get the command description, I execute that command, I see what this command does.
Help Text	2.4	The game provides a "HELP" command for accessing the help text. Upon entering the "HELP" command, the game immediately displays the help text. The help text is presented in a clear and easily readable format, ensuring the player's understanding of available commands and game mechanics. The player can exit the help text display seamlessly at any point, returning to the game interface without any hindrances.
Clues Inventory	2.7	Given that I am a player in the game, When I come across a clue during my exploration, then my expectation is that the clue is automatically stored in my clue inventory. Given that I am a player in the game, When I access my clue inventory, then my expectation is that the location of clues will also be displayed in order to make a more informed guess.
Suspect List	2.9	Given that I am a player in the game, when I access the suspect list, then my expectation is to see a comprehensive list of all the available suspects in the murder mystery, including their names, descriptions, backgrounds, and any relevant information that might make them potential suspects. Given that I am a player in the game, when I successfully identify the killer or make my final guess, then my expectation is that the suspect list provides an outcome for each suspect, indicating their innocence or guilt.

Time Box	3.5	Given I am a player in a time-constrained game, When I start or resume the game, Then I expect to see a timer prominently displayed on the screen. Given the game has an initial time limit set, When I start the game, Then I expect the timer to begin counting down from the initial time limit. Given I am actively playing the game, When I glance at the timer, Then I expect the timer to accurately display the remaining time, updating in real-time. Given the game has a time limit, and the timer is visible, When the timer reaches 0:00 (time limit expiration), Then I expect a clear notification or message to appear, indicating that the game has concluded due to time expiration.
Saving Progress	3.8	Given that I am a player in the game, when I choose to save my progress, then my expectation is that the game allows me to create multiple save slots. Given that I am a player in the game, when I save my progress, then my expectation is that the game captures and stores critical game states, including the player's location, inventory contents, and collected clues, ensuring a seamless continuation of the game experience. Given that I am a player in the game, when I load a saved game, then my expectation is that the game accurately restores the saved state, placing me exactly where I left off with all in-game progress intact.

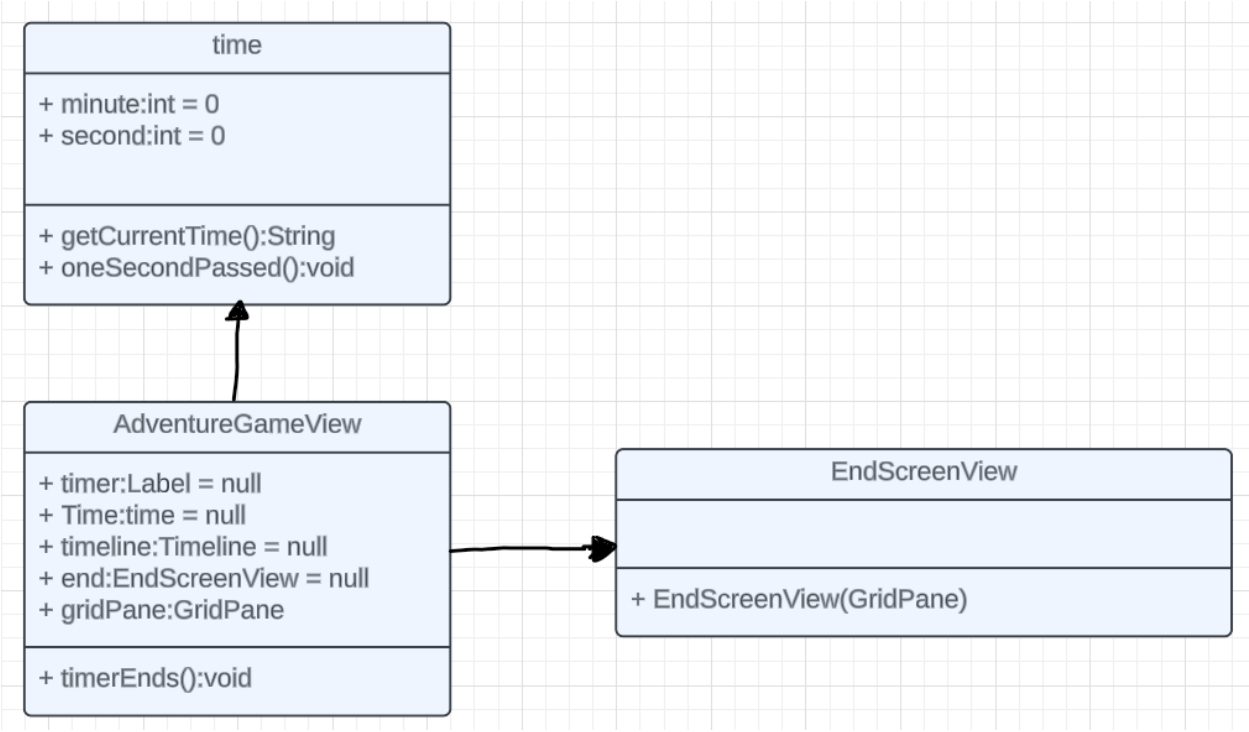
SECTION 3: SOFTWARE DESIGN

This section should describe the design patterns you will use to realize your user stories. For each design pattern, provide a draft of UML diagrams to show how you'll implement the pattern as well as a written description that details how it will be used in the context of your user stories. An example of a pattern description is provided below. Note that you can adjust these patterns as you progress with your implementation, as this is necessary.

Design Pattern #1: Observer Pattern

Overview: This pattern will be used to implement Timer and TimerEnds.

UML Diagram:



Implementation Details: The UML diagram outlines these main components:

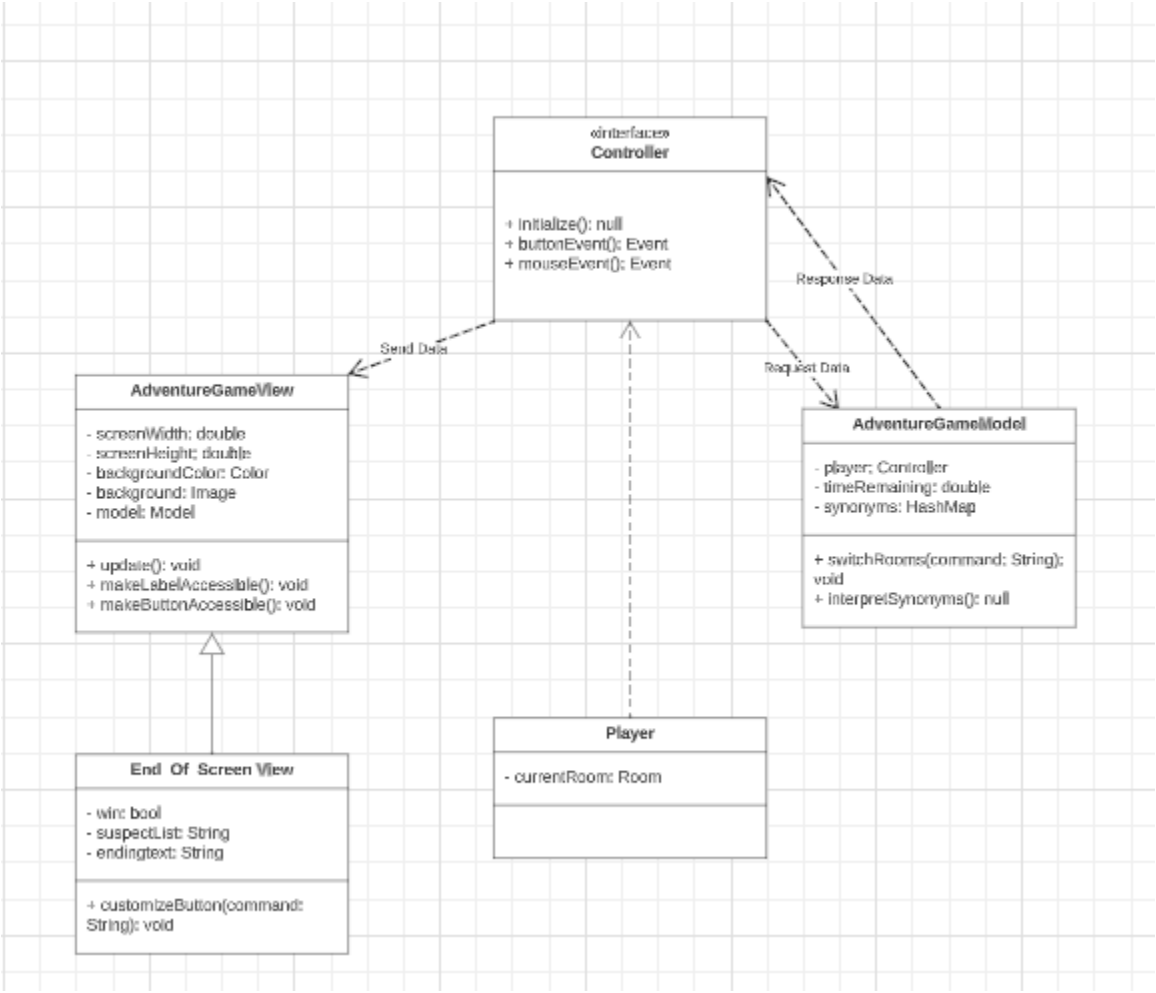
- The *time* class, which contains *minute* and *second* as attributes, *getCurrentTime* and *oneSecondPassed* methods.
- The *AdventureGameView* class, which contains *timer, Time, timeline, end,* and *gridPane* as attributes. This class also contains the *timerEnds* method.
- The *EndScreenView* class, which contains the *EndScreenView* method that takes a *GridPane* as a parameter.

The *time* attribute here acts as the observer of the time passed during the game. The *timer* attribute contains the string value of *time* which displays it to the user. While the game runs, *timeline* constantly updates *time* and *timer* every second passed. If *time* has not reached 0 *minutes* and 0 *seconds,* it runs the *oneSecondPassed* method. Otherwise, it runs *timerEnds*. Here, *timerEnds* acts as the notifier to all *children* in *gridPane* if the timer ends. When *timerEnds* runs, *the EndScreenView* method runs and edits every *children* in *gridPane* so that the end screen pops up.

Design Pattern #2: Pattern

Overview: This pattern will be used for the player to visit different rooms

UML Diagram:



Implementation Details: The UML diagram outlines these main components:

- The *AdventureGameModel* class, which includes two methods: *retireveData* and *initialize*. The *GameModel* class is what will handle most of the background processes.
- The *Player* class, with additional attributes *currentRoom* and *inventory*, as well as methods like *takeClue* and *dropClue.* The implementation of these methods will rely heavily on the observer pattern.
- The *AdventureGameView* class, which is the main view class and contains standard attributes required for GUI based objects, as well as additional methods for ensuring accessibility.
- The *End_oF_Screen_View,* each of which has unique functionality that the player will require to use, in order to navigate the application properly.

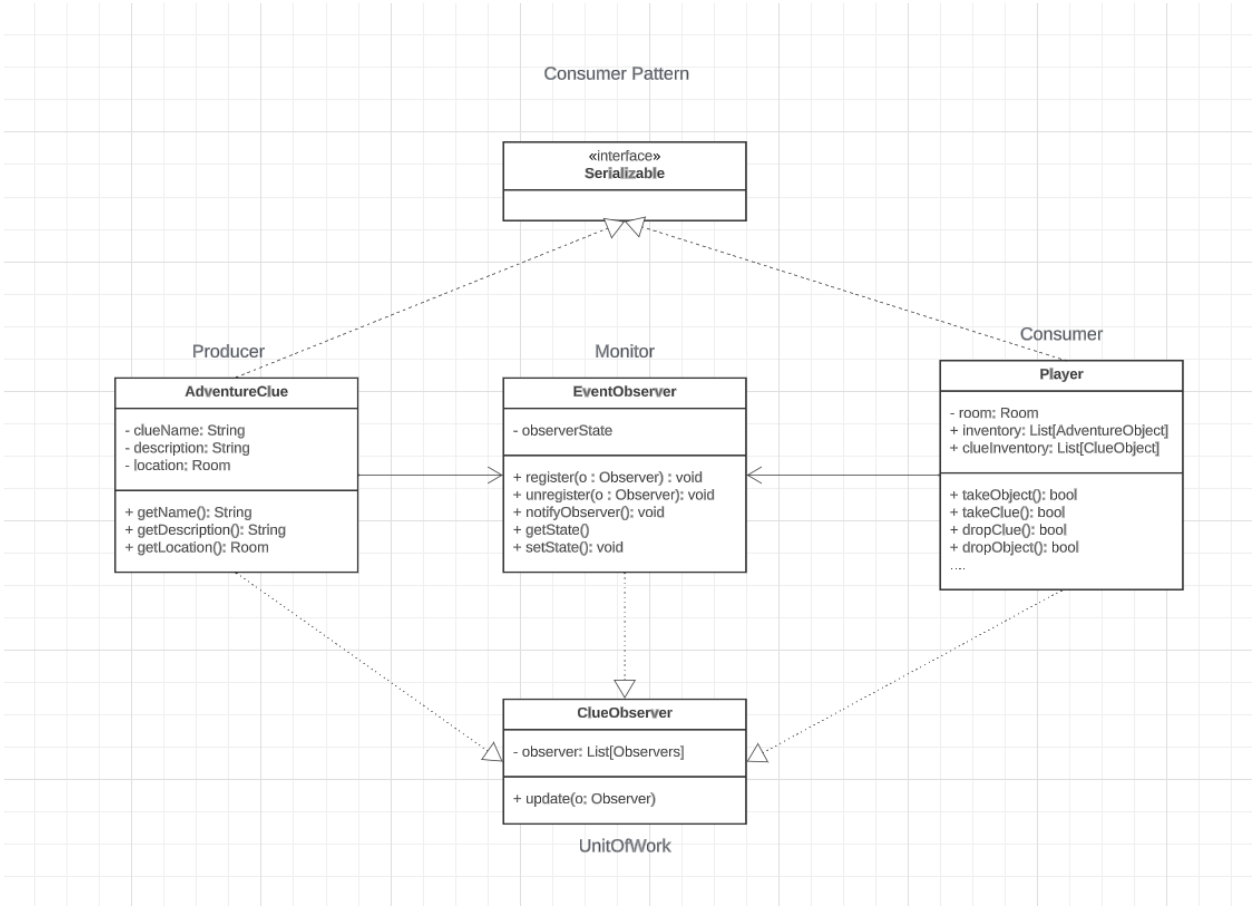
The *View* class contains the sub-classes and objects that are responsible for displaying the user interface. The *Model* section contains the sub-classes and objects that handle background operations. The controller interface contains implementations that handle user input and manage the flow of the application.

The program will start by running the *retrieveData* method in the Model class to initialize the model used. The user's *View* will be set to the *MainMenuView.* The views will update, based on the events that the player chooses to do (handled by the *Controller*), as well as the data stored in the *Model.*

The *interpretSynonyms* method in the *GameModel* class, will allow the program to recognize certain synonyms that are used for common commands. The *GameOverView* will handle events related to the game ending. The *makeButtonAccessible* and *makeLabelAccessible* methods in the *View* interface will handle accessibility concerns. The *buttonEventHandler* and *mouseEventHandler* will handle the taking and dropping of clues. Therefore, all user stories are handled with this implementation of the MVC Pattern.

Design Pattern #3: Consumer Pattern

Overview: This pattern will be used to implement the updating functionality clues in the inventory.



Implementation Details: The UML diagram outlines these main components:

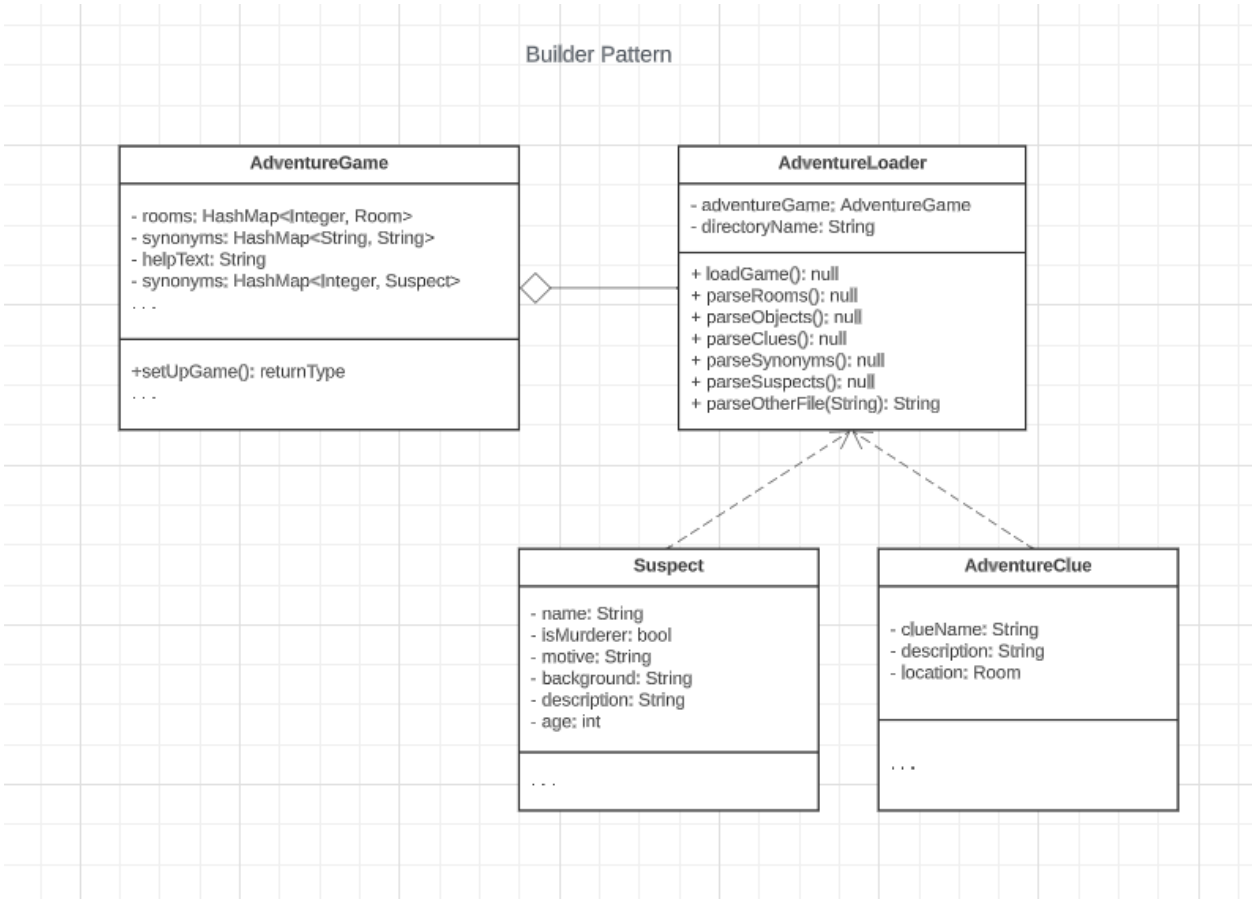
- An *EventObserver* abstract class that any observer class can inherit.
- An *AdventureClue* class acts as a producer and produces new clues objects.
- A *Player* class that acts as a consumer and is updated through the *EventObserver* when changes are observed to any clues.
- A *ClueObserver* that observes all updates to clues and notifies all observers.

Every executable event involving clues will be passed through *ClueObserver*. These events then notify all registered observers (*Player's*) through a *notifyObservers* method in the *EventObserver* class. To register as an observer, the *register* method can be used by passing in the specific observer, and to unregister, the same steps would be followed but with the *unregister* method.

Design Pattern #4: Builder Pattern

Overview: This pattern was used to simplify the set up for the adventure game.

UML Diagram:



Implementation Details: The UML diagram outlines these main components:

- The AdventureGame which acts as the Director of the builder pattern, with the method setUpGame() being synonymous to the construct() method of the Director class.
- The AdventureLoader acts as the ConcreteBuilder, which builds all necessary attributes of the AdventureGame. (Note: the abstract Builder interface was omitted due to the small scale of the project)
- Suspect and AdventureClue, which act as Product classes, that are created from the AdventureLoader.

The program will start by running the *setUpGame* method in the *AdventureModel* class to initialize the model used. *setUpGame* consists mostly of the *loadGame* method in the *AdventureLoader* class, which builds the products used in AdventureGame. The products built by *AdventureLoader* include *Suspect* and *AdventureClue* which become become essential to the program later on.

Tasks for Sprint #1: (Implementing the structure of the game/enhancements from A2)

- Tasks to implement:
    - Sehaj
        - VisualizingRoom (1.1)
            - Create files for the new rooms that are related to the game.
    - Wahib
        - RoomConnection (1.3)
            - Remade the entire rooms.txt, file, with different room names, descriptions, etc. Also replaces room images to match the rooms.
    - Kevin
        - VisualizingObject (1.2)
            - Create a new files for new objects that are related to the game
    - Reto
        - Timer (1.4)
            - Create a time class that keeps track of how much time is left. Then make a time object and a Timeline object inside AdventureGameView to update the time object every second. Then make a Label to inform the user how much time is left for them.

Tasks for Sprint #2: (Implementing the structure of the game/enhancements from A2)

- Tasks to implement:
    - Sehaj
        - CluesFeature (2.1)
            - Create a new AdventureClue interface similar to AdventureObject, and update clues in both the inventory and objects/clues in room as the user makes their way through the game by getting updates from an observer.
    - Wahib
        - SuspectFeature (2.4)
            - Create a suspects.txt and updated AdventureLoader.java and AdventureGame.java to handle the parsing of the text file.
    - Kevin
        - End_Of_Game (2.3)
            - Create an End_Screen_View class with suspectList and endingText attributes where it will display the right text based on the users' choice of killer in the game
    - Reto
        - TimerEnds (2.2)
            - Show the end game screen when the timer ends by using the method Christian created in his End_Of_Game implementation.

Tasks for Sprint #3: (Implementing aids)

- Tasks to implement:
    - Sehaj
        - ColourInverted (3.1)
            - Create an End_Screen_View class with suspectList and endingText attributes where it will display the right text based on the users' choice of killer in the game
    - Wahib
        - ContrastRatio (3.4)
            - Ensure that all labels, images and backgrounds maintain 4:5:1 ratio upon instantiation and updates.
    - Kevin
        - Map (3.3)
            - Make an image of the location for each room and show it up in the GUI by adding an ImageView to gridPane.
    - Reto
        - Timer (3.2)
            - Use the Free Text-to-Speech library to read the room description. Implement this by replacing the current ArticulateRoomDescription inside AdventureGameView.