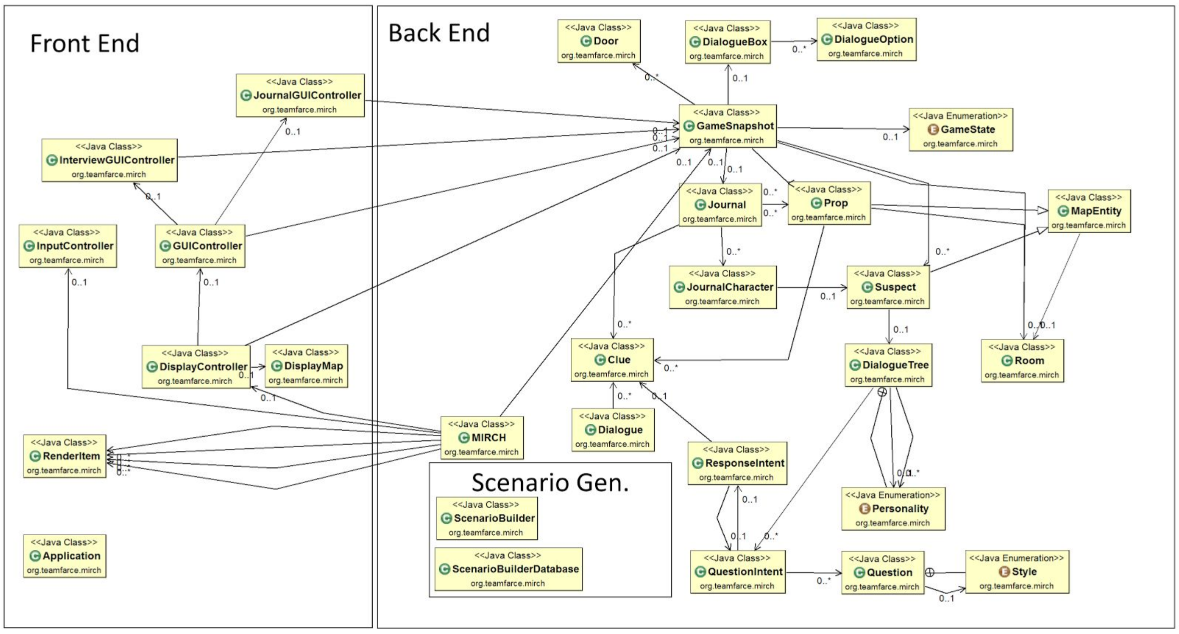
Architecture report

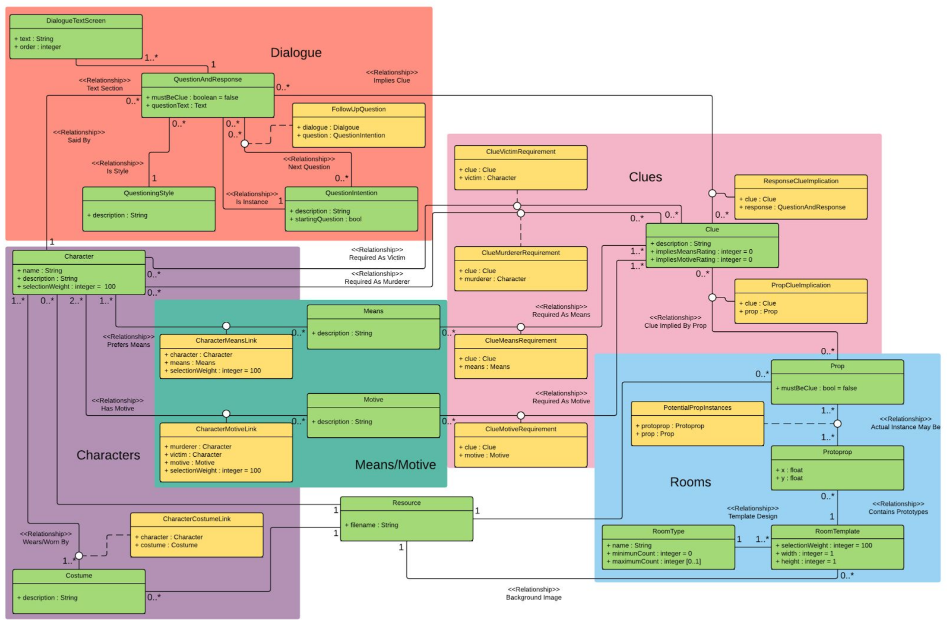
The diagram represents the concrete architecture of the team's project in UML. The code is split into a two part structure, the front end and the back end. The team felt that UML was appropriate as the language for the architecture because it allows visualisation of the program with an increased level of detail, improving the traceability of the program. The team followed this model as it was helpful in accommodating for the parallel development of for the two parts of the program. *(The diagrams are quite big and full resolution versions can be found on the website)*

<u>Overall UML Diagram: Link at the end of the document</u>



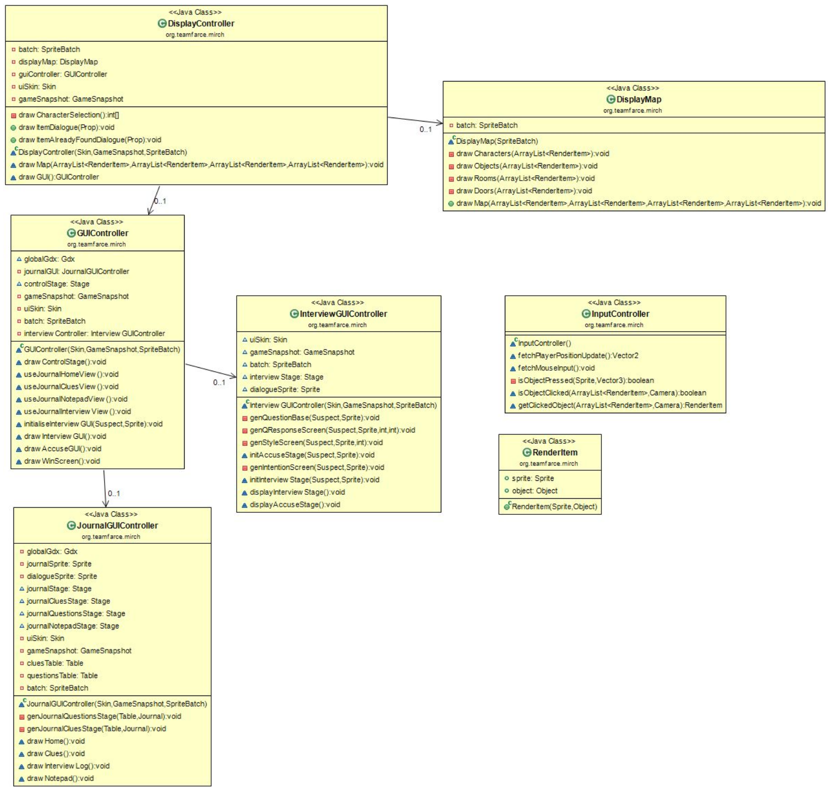
Front End
Back End
Scenario Gen.
<<Java Class>> JournalGUIController org.teamfarce.mirch
<<Java Class>> InterviewGUIController org.teamfarce.mirch
<<Java Class>> InputController org.teamfarce.mirch
<<Java Class>> GUIController org.teamfarce.mirch
<<Java Class>> DisplayController org.teamfarce.mirch
<<Java Class>> DisplayMap org.teamfarce.mirch
<<Java Class>> RenderItem org.teamfarce.mirch
<<Java Class>> Application org.teamfarce.mirch
<<Java Class>> Door org.teamfarce.mirch
<<Java Class>> DialogueBox org.teamfarce.mirch
<<Java Class>> DialogueOption org.teamfarce.mirch
<<Java Class>> GameSnapshot org.teamfarce.mirch
<<Java Enumeration>> GameState org.teamfarce.mirch
<<Java Class>> Journal org.teamfarce.mirch
<<Java Class>> Prop org.teamfarce.mirch
<<Java Class>> MapEntity org.teamfarce.mirch
<<Java Class>> JournalCharacter org.teamfarce.mirch
<<Java Class>> Suspect org.teamfarce.mirch
<<Java Class>> Clue org.teamfarce.mirch
<<Java Class>> DialogueTree org.teamfarce.mirch
<<Java Class>> Room org.teamfarce.mirch
<<Java Class>> MIRCH org.teamfarce.mirch
<<Java Class>> Dialogue org.teamfarce.mirch
<<Java Class>> ResponseIntent org.teamfarce.mirch
<<Java Enumeration>> Personality org.teamfarce.mirch
<<Java Class>> ScenarioBuilder org.teamfarce.mirch
<<Java Class>> ScenarioBuilderDatabase org.teamfarce.mirch
<<Java Class>> QuestionIntent org.teamfarce.mirch
<<Java Class>> Question org.teamfarce.mirch
<<Java Enumeration>> Style org.teamfarce.mirch

<u>Scenario Builder Diagram: Link at the end of the document</u>



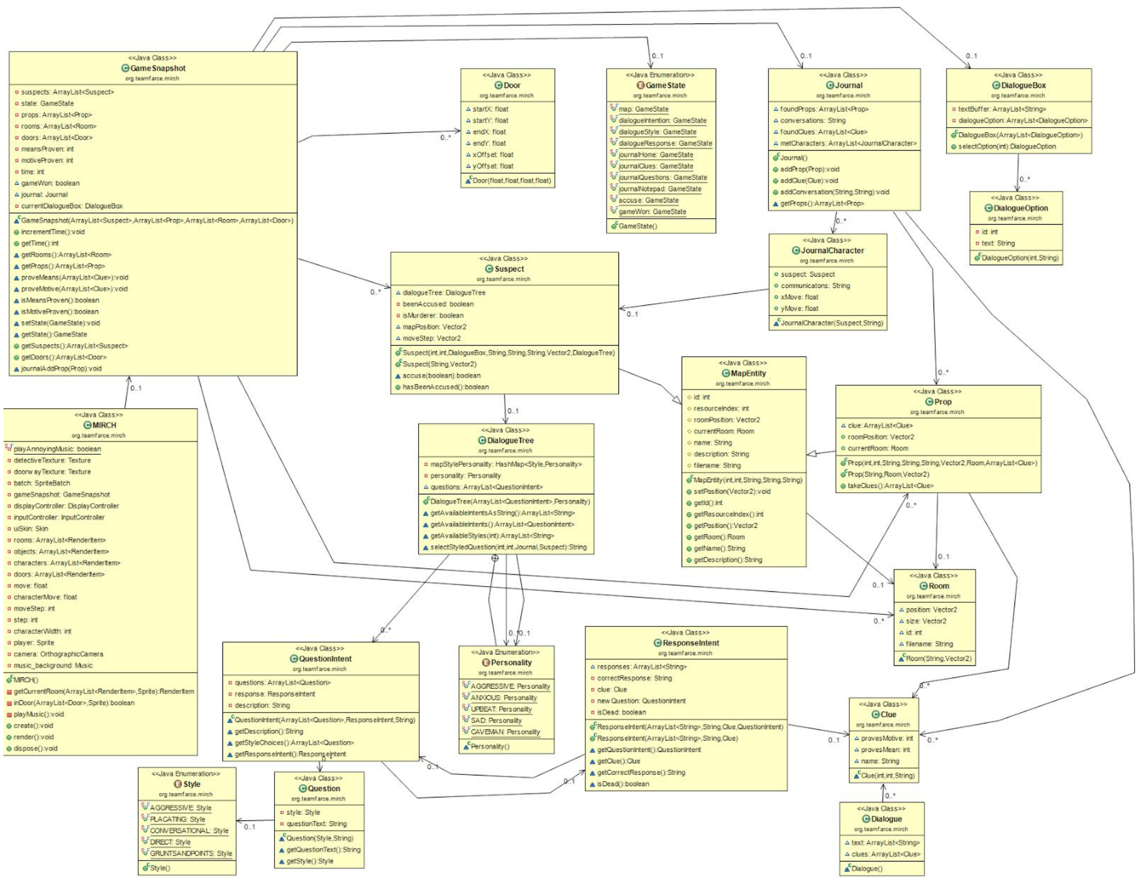
Dialogue
DialogueTextScreen + text : String + order : integer
QuestionAndResponse + mustBeClue : boolean = false + questionText : Text
FollowUpQuestion + dialogue : Dialgoue + question : QuestionIntention
QuestioningStyle + description : String
QuestionIntention + description : String + startingQuestion : bool
<<Relationship>> Text Section
<<Relationship>> Said By
<<Relationship>> Is Style
<<Relationship>> Next Question
<<Relationship>> Is Instance
<<Relationship>> Implies Clue
Characters
Character + name : String + description : String + selectionWeight : integer = 100
<<Relationship>> Required As Victim
<<Relationship>> Required As Murderer
<<Relationship>> Prefers Means
<<Relationship>> Has Motive
<<Relationship>> Wears/Worn By
CharacterCostumeLink + character : Character + costume : Costume
Costume + description : String
Means/Motive
Means + description : String
Motive + description : String
CharacterMeansLink + character : Character + means : Means + selectionWeight : integer = 100
CharacterMotiveLink + murderer : Character + victim : Character + motive : Motive + selectionWeight : integer = 100
Resource + filename : String
<<Relationship>> Background Image
Clues
ClueVictimRequirement + clue : Clue + victim : Character
ClueMurdererRequirement + clue : Clue + murderer : Character
ClueMeansRequirement + clue : Clue + means : Means
ClueMotiveRequirement + clue : Clue + motive : Motive
Clue + description : String + impliesMeansRating : integer = 0 + impliesMotiveRating : integer = 0
ResponseClueImplication + clue : Clue + response : QuestionAndResponse
PropClueImplication + clue : Clue + prop : Prop
<<Relationship>> Required As Means
<<Relationship>> Required As Motive
<<Relationship>> Clue Implied By Prop
Rooms
Prop + mustBeClue : bool = false
PotentialPropInstances + protoprop : Protoprop + prop : Prop
Protoprop + x : float + y : float
<<Relationship>> Actual Instance May Be
<<Relationship>> Template Design
<<Relationship>> Contains Prototypes
RoomType + name : String + minimunCount : integer = 0 + maximumCount : integer [0..1]
RoomTemplate + selectionWeight : integer = 100 + width : integer = 1 + height : integer = 1

Detailed Front End Diagram: Link at the end of the document



<<Java Class>>
DisplayController
org.teamfarce.mirch
batch: SpriteBatch
displayMap: DisplayMap
guiController: GUIController
uiSkin: Skin
gameSnapshot: GameSnapshot
drawCharacterSelection():int[]
drawItemDialogue(Prop):void
drawItemAlreadyFoundDialogue(Prop):void
DisplayController(Skin,GameSnapshot,SpriteBatch)
drawMap(ArrayList<RenderItem>,ArrayList<RenderItem>,ArrayList<RenderItem>,ArrayList<RenderItem>):void
drawGUI():GUIController
0..1
<<Java Class>>
DisplayMap
org.teamfarce.mirch
batch: SpriteBatch
DisplayMap(SpriteBatch)
drawCharacters(ArrayList<RenderItem>):void
drawObjects(ArrayList<RenderItem>):void
drawRooms(ArrayList<RenderItem>):void
drawDoors(ArrayList<RenderItem>):void
drawMap(ArrayList<RenderItem>,ArrayList<RenderItem>,ArrayList<RenderItem>,ArrayList<RenderItem>):void
0..1
<<Java Class>>
GUIController
org.teamfarce.mirch
globalGdx: Gdx
journalGUI: JournalGUIController
controlStage: Stage
gameSnapshot: GameSnapshot
uiSkin: Skin
batch: SpriteBatch
interviewController: InterviewGUIController
GUIController(Skin,GameSnapshot,SpriteBatch)
drawControlStage():void
useJournalHomeView():void
useJournalCluesView():void
useJournalNotepadView():void
useJournalInterviewView():void
initialiseInterviewGUI(Suspect,Sprite):void
drawInterviewGUI():void
drawAccuseGUI():void
drawWinScreen():void
0..1
<<Java Class>>
InterviewGUIController
org.teamfarce.mirch
uiSkin: Skin
gameSnapshot: GameSnapshot
batch: SpriteBatch
interviewStage: Stage
dialogueSprite: Sprite
InterviewGUIController(Skin,GameSnapshot,SpriteBatch)
genQuestionBase(Suspect,Sprite):void
genQResponseScreen(Suspect,Sprite,int,int):void
genStyleScreen(Suspect,Sprite,int):void
initAccuseStage(Suspect,Sprite):void
genIntentionScreen(Suspect,Sprite):void
initInterviewStage(Suspect,Sprite):void
displayInterviewStage():void
displayAccuseStage():void
<<Java Class>>
InputController
org.teamfarce.mirch
InputController()
fetchPlayerPositionUpdate():Vector2
fetchMouseInput():void
isObjectPressed(Sprite,Vector3):boolean
isObjectClicked(ArrayList<RenderItem>,Camera):boolean
getClickedObject(ArrayList<RenderItem>,Camera):RenderItem
<<Java Class>>
RenderItem
org.teamfarce.mirch
sprite: Sprite
object: Object
RenderItem(Sprite,Object)
0..1
<<Java Class>>
JournalGUIController
org.teamfarce.mirch
globalGdx: Gdx
journalSprite: Sprite
dialogueSprite: Sprite
journalStage: Stage
journalCluesStage: Stage
journalQuestionsStage: Stage
journalNotepadStage: Stage
uiSkin: Skin
gameSnapshot: GameSnapshot
cluesTable: Table
questionsTable: Table
batch: SpriteBatch
JournalGUIController(Skin,GameSnapshot,SpriteBatch)
genJournalQuestionsStage(Table,Journal):void
genJournalCluesStage(Table,Journal):void
drawHome():void
drawClues():void
drawInterviewLog():void
drawNotepad():void

Detailed Back End Diagram: Link at the end of the document



Justification of Concrete Architecture

For the sake of speeding up development of the project as well as for streamlining the process of quality control, modules of the game were moved around between the two parts in the project. Instead of the back-end using two interfaces (outputter and inputter) to communicate with the front-end, it now uses a class for finding which state the game is in, and a class which also contains the logic for the game for each loop. The architecture we had originally intended to use had to be changed slightly once we had begun due to the way we had envisioned the game engine working was slightly too simplistic for what was actually required in the requirements.

The architecture still works using a two-part structure consisting of a front-end, which controls the inputs and draws the game to the display, and a back-end to handle the scenario generation, game loop and the game state.

We have managed to keep with our concept of using a modularised approach for our program structure. As detailed below, we have split most of the front-end into separate detailed classes to allow easy parallel programming between the team and to ensure that the code was easily maintainable by not just our own team but by any future team that may work on our code.

This implementation of architecture would allow us to test multiple front-ends with the back-end allowing different rendering options and control schemes to be implemented. These different front-ends could also be written in different programming languages if required to do so.

The Front End

The front end now performs the following processes:

- Control the initialisation of the back-end, as well as its termination on the closing of the GUI - therefore also disposing of the assets used from memory.
- Displays the GUI and the map, along with all the characters and props
- Collect inputs from the user, converts them to meaningful commands and passes these to the backend; in addition to collecting outputs generated by the backend and displaying these onto the user's display.
- Handles movement of NPCs and the player.

The most prominent change from the abstract architecture is the handling of movement from the player and the NPCs which was changed since the team decided there was no need to add complexity to the back end about the position of the sprite (the requirement of moving NPCs remained unaffected by this change).

The task of game rendering and input translating from the abstract architecture for the front end was factored into multiple classes (DisplayController, GUIController, InputController, etc.) for the purpose of data encapsulation to allow different team members to work on the game rendering concurrently.

The Back End

The back end performs the following processes:

- Collect information and send back any relevant information back to the front-end.
- Taking the generated game state from the scenario generator and checking whether the means and motive has been proven.
- Initializes and updates the game state in each loop multiple times in a second
- Initializes the scenario generation
- Calculates the score at the end of the game

The GameSnapshot class keeps track of the game state, which includes all the rooms, props, clues, etc.) after the initialization from the game scenario (ScenarioBuilder) through every game loop while the MIRCH class handles the updates and game logic (acting as the game controller from the abstract architecture).

Collecting the game logic in the back end (in the MIRCH class) allows the potential implementation of multiple front ends making requirements for visual features independent of the requirements for the game logic (core requirements), therefore making any future requirement changes easier to accommodate.

The Scenario Generation

The game scenario generator uses multiple different classes to calculate the initial game setup including the murderer and clues; picks the suspects to be used within the scenario; provide questions and answers for the interrogation of characters;

To do this, the generator looks at a number of different variables from a database to generate links between clues and suspects including dialogue trees that the player may come across during gameplay. A “murderer-victim-motive link” list is then made to allow two characters to be chosen to be placed in the game scenario. These characters will relate to the Murderer and the Victim with a motive being put in place that the player will be able to discover. From this, clues can be generated with some relating to the murderer, some not to throw the player off the scent and to add another layer of complexity to the game. This is then fed into the GameSnapshot as the initial state for the game.

Managing the scenario in such a way allowed team members to work on creating a database of rooms,characters,props etc. out of which a random selection is created, thus fulfilling the dynamic requirements of the game.

Updated Requirements Link:
https://teamfarce.github.io/MIRCH/designdocs/submit_2/updated/Req2.pdf
Abstract Architecture Link:
https://teamfarce.github.io/MIRCH/designdocs/submit_1/Arch1.pdf
Class Diagram Links:
https://teamfarce.github.io/MIRCH/designdocs/submit_2/new/Class_diagram.png
https://teamfarce.github.io/MIRCH/designdocs/submit_2/new/back_end.png
https://teamfarce.github.io/MIRCH/designdocs/submit_2/new/Front_end.png
https://teamfarce.github.io/MIRCH/designdocs/submit_2/new/DataStoreDiagram.png