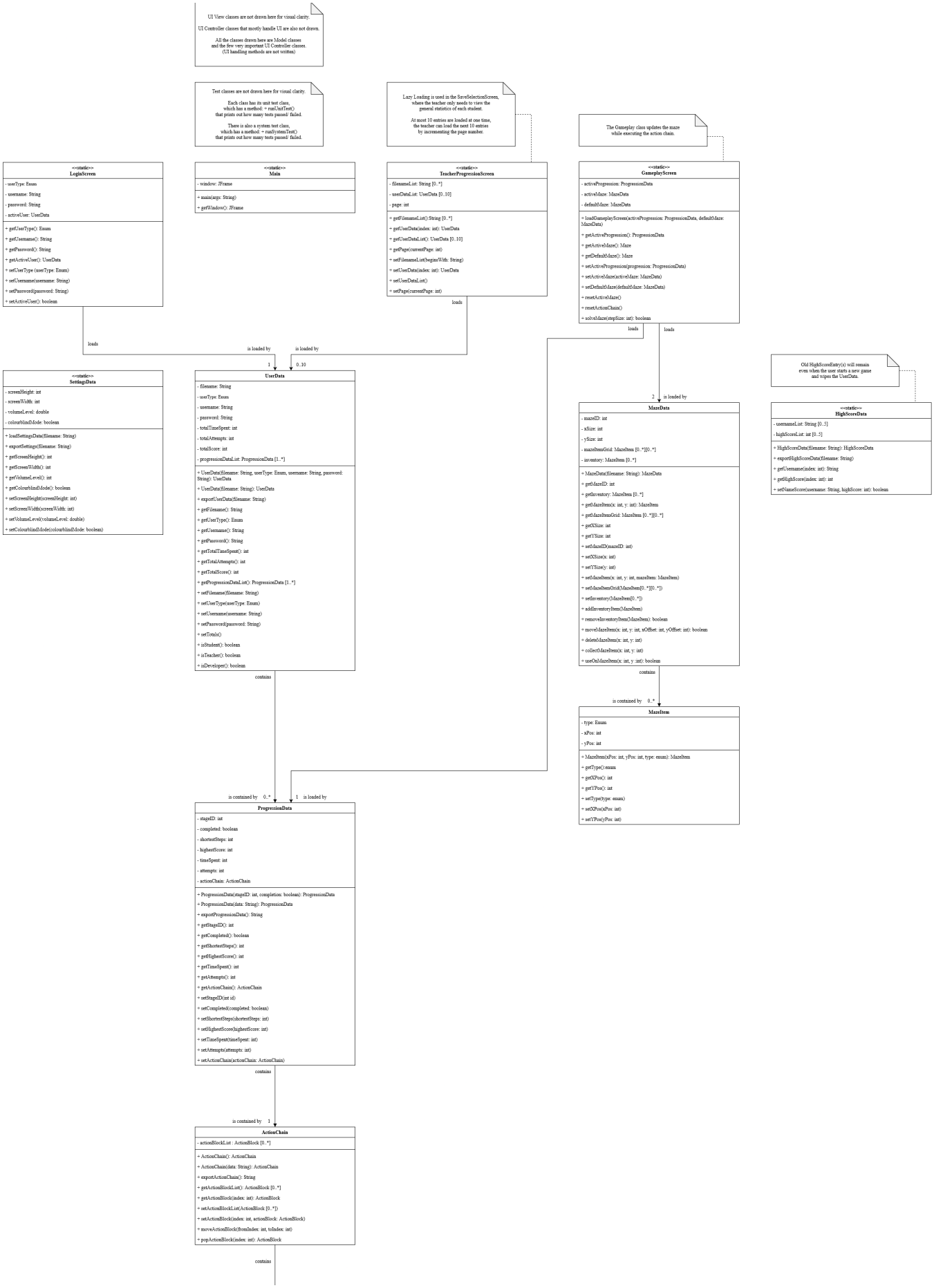
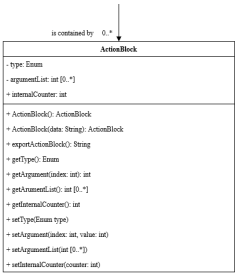
Class Diagrams

Table of Contents

Class Diagram(s)

Download Link: https://drive.google.com/file/d/1PAgSX39gpMdv9PpoJh8YVaIGUhchYtj4/view?usp=sharing


UI View classes are not drawn here for visual clarity.
UI Controller classes that mostly handle UI are also not drawn.
All the classes drawn here are Model classes
and the few very important UI Controller classes.
(UI handling methods are not written)
Test classes are not drawn here for visual clarity.
Each class has its unit test class,
which has a method: + runUnitTest()
that prints out how many tests passed/ failed.
There is also a system test class,
which has a method: + runSystemTest()
that prints out how many tests passed/ failed.
Lazy Loading is used in the SaveSelectionScreen,
where the teacher only needs to view the
general statistics of each student.
At most 10 entries are loaded at one time,
the teacher can load the next 10 entries
by incrementing the page number.
The Gameplay class updates the maze
while executing the action chain.
Old HighScoreEntry(s) will remain
even when the user starts a new game
and wipes the UserData.
<<static>>
LoginScreen
- userType: Enum
- username: String
- password: String
- activeUser: UserData
+ getUserType(): Enum
+ getUsername(): String
+ getPassword(): String
+ getActiveUser(): UserData
+ setUserType (userType: Enum)
+ setUsername(username: String)
+ setPassword(password: String)
+ setActiveUser(): boolean
<<static>>
Main
- window: JFrame
+ main(args: String)
+ getWindow(): JFrame
<<static>>
TeacherProgressionScreen
- filenameList: String [0..*]
- userDataList: UserData [0..10]
- page: int
+ getFilenameList():String [0..*]
+ getUserData(index: int): UserData
+ getUserDataList(): UserData [0..10]
+ getPage(currentPage: int)
+ setFilenameList(beginsWith: String)
+ setUserData(index: int): UserData
+ setUserDataList()
+ setPage(currentPage: int)
<<static>>
GameplayScreen
- activeProgression: ProgressionData
- activeMaze: MazeData
- defaultMaze: MazeData
+ loadGameplayScreen(activeProgression: ProgressionData, defaultMaze: MazeData)
+ getActiveProgression(): ProgressionData
+ getActiveMaze(): Maze
+ getDefaultMaze(): Maze
+ setActiveProgression(progression: ProgressionData)
+ setActiveMaze(activeMaze: MazeData)
+ setDefaultMaze(defaultMaze: MazeData)
+ resetActiveMaze()
+ resetActionChain()
+ solveMaze(stepSize: int): boolean
<<static>>
SettingsData
- screenHeight: int
- screenWidth: int
- volumeLevel: double
- colourblindMode: boolean
+ loadSettingsData(filename: String)
+ exportSettings(filename: String)
+ getScreenHeight(): int
+ getScreenWidth(): int
+ getVolumeLevel(): int
+ getColourblindMode(): boolean
+ setScreenHeight(screenHeight: int)
+ setScreenWidth(screenWidth: int)
+ setVolumeLevel(volumeLevel: double)
+ setColourblindMode(colourblindMode: boolean)
UserData
- filename: String
- userType: Enum
- username: String
- password: String
- totalTimeSpent: int
- totalAttempts: int
- totalScore: int
- progressionDataList: ProgressionData [1..*]
+ UserData(filename: String, userType: Enum, username: String, password: String): UserData
+ UserData(filename: String): UserData
+ exportUserData(filename: String)
+ getFilename(): String
+ getUserType(): Enum
+ getUsername(): String
+ getPassword(): String
+ getTotalTimeSpent(): int
+ getTotalAttempts(): int
+ getTotalScore(): int
+ getProgressionDataList(): ProgressionData [1..*]
+ setFilename(filename: String)
+ setUserType(userType: Enum)
+ setUsername(username: String)
+ setPassword(password: String)
+ setTotals()
+ isStudent(): boolean
+ isTeacher(): boolean
+ isDeveloper(): boolean
MazeData
- mazeID: int
- xSize: int
- ySize: int
- mazeItemGrid: MazeItem [0..*][0..*]
- inventory: MazeItem [0..*]
+ MazeData(filename: String): MazeData
+ getMazeID: int
+ getInventory: MazeItem [0..*]
+ getMazeItem(x: int, y: int): MazeItem
+ getMazeItemGrid: MazeItem [0..*][0..*]
+ getXSize: int
+ getYSize: int
+ setMazeID(mazeID: int)
+ setXSize(x: int)
+ setYSize(y: int)
+ setMazeItem(x: int, y: int, mazeItem: MazeItem)
+ setMazeItemGrid(MazeItem[0..*][0..*])
+ setInventory(MazeItem[0..*])
+ addInventoryItem(MazeItem)
+ removeInventoryItem(MazeItem): boolean
+ moveMazeItem(x: int, y: int, xOffset: int, yOffset: int): boolean
+ deleteMazeItem(x: int, y: int)
+ collectMazeItem(x: int, y: int)
+ useOnMazeItem(x: int, y :int): boolean
<<static>>
HighScoreData
- usernameList: String [0..5]
- highScoreList: int [0..5]
+ HighScoreData(filename: String): HighScoreData
+ exportHighScoreData(filename: String)
+ getUsername(index: int): String
+ getHighScore(index: int): int
+ setNameScore(username: String, highScore: int): boolean
MazeItem
- type: Enum
- xPos: int
- yPos: int
+ MazeItem(xPos: int, yPos: int, type: enum): MazeItem
+ getType():enum
+ getXPos(): int
+ getYPos(): int
+ setType(type: enum)
+ setXPos(xPos: int)
+ setYPos(yPos: int)
ProgressionData
- stageID: int
- completed: boolean
- shortestSteps: int
- highestScore: int
- timeSpent: int
- attempts: int
- actionChain: ActionChain
+ ProgressionData(stageID: int, completion: boolean): ProgressionData
+ ProgressionData(data: String): ProgressionData
+ exportProgressionData(): String
+ getStageID(): int
+ getCompleted(): boolean
+ getShortestSteps(): int
+ getHighestScore(): int
+ getTimeSpent(): int
+ getAttempts(): int
+ getActionChain(): ActionChain
+ setStageID(int id)
+ setCompleted(completed: boolean)
+ setShortestSteps(shortestSteps: int)
+ setHighestScore(highestScore: int)
+ setTimeSpent(timeSpent: int)
+ setAttempts(attempts: int)
+ setActionChain(actionChain: ActionChain)
ActionChain
- actionBlockList : ActionBlock [0..*]
+ ActionChain(): ActionChain
+ ActionChain(data: String): ActionChain
+ exportActionChain(): String
+ getActionBlockList(): ActionBlock [0..*]
+ getActionBlock(index: int): ActionBlock
+ setActionBlockList(ActionBlock [0..*])
+ setActionBlock(index: int, actionBlock: ActionBlock)
+ moveActionBlock(fromIndex: int, toIndex: int)
+ popActionBlock(index: int): ActionBlock
loads
loads
is loaded by
1
is loaded by
0..10
loads
loads
2
is loaded by
contains
is contained by
0..*
contains
is contained by
0..*
1
is loaded by
contains
is contained by
1
contains



Textual Description(s)/ CRC Card(s)

The classes are designed based on the View-Model-Controller pattern. For visual clarity, all of the View classes (UI) and some of the Controller classes (mostly UI handling) are not drawn here. Only the Model classes and some important Controller classes are drawn here. The UI-related methods in the important Controller classes are also not drawn here.

Main

Class: Main	
This controller class is the starting point of the program. It creates the game window and opens the login screen.	
Responsibility	**Collaborator**
Loads the game settings.	JWindow (JavaX Swing)
Creates and store the game window.	LoginScreen
Opens the login screen.	

LoginScreen

Class: LoginScreen	
This controller class stores, updates, and checks the current login data provided by LoginScreenUI.	
Responsibility	**Collaborator**
Tracks and updates the user type, username, and password.	LoginScreenUI (not drawn)
Validates user password.	UserData
Loads/ Creates a UserData object.	

SaveSelectionScreen

Class: TeacherProgressionScreen	
This controller class loads a list of UserData and updates the TeacherProgressionScreenUI.	
Responsibility	**Collaborator**
Tracks and Updates the list of UserData currently loaded.	TeacherProgressionScreenUI (not drawn)
Updates the TeacherProgressionScreen.	UserData

GameplayScreen

Class: GameplayScreen	
This controller class interprets the ActionChain, updates MazeData in real time, and then updates the GameplayScreenUI.	
Responsibility	**Collaborator**
Tracks and updates the ProgressionData loaded.	GameplayScreenUI (not drawn)
Load the default maze layout.	ProgressionData

Updates/ Resets the active maze layout.	MazeData
Updates the GameplayScreenUI.	
Interprets the ActionChain in real time, and executes the instructions.	

SettingsData

Class: SettingsData	
This model class stores and updates the game's settings. It loads/ saves data using the settings data file. All users share the same game settings.	
Responsibility	**Collaborator**
Loads/ Updates the settings file.	SettingsScreen (not drawn)

UserData

Class: UserData	
This model class stores a user's progress in the game. It loads/ saves data using the user's data file. The user's ProgressionData (per stage) is stored as separate objects.	
Responsibility	**Collaborator**
Loads/ Updates the user data file.	LoginScreen
Stores/ Updates the filename, user type, username, password, and other general player statistics.	TeacherProgressionScreen
	ProgressionData

ProgressionData

Class: ProgressionData	
This model class stores a user's progress in a stage. It loads/ saves data by communicating with the UserData object. The ActionChain is stored as a separate object.	
Responsibility	**Collaborator**
Loads/ Exports the per-stage progression data.	UserData
Stores/ Updates the stage ID, completion, and other detailed per-stage player statistics.	ActionChain
	TeacherProgressionScreen

ActionChain

Class: ActionChain	
This model class stores a user's saved action chain in a stage. It loads/ saves data by communicating with the ProgressionData object. Each ActionBlock is stored as a separate object.	
Responsibility	**Collaborator**
Loads/ Exports the per-stage action chain data.	ProgressionData
Stores/ Updates the list of action blocks.	ActionBlock

ActionBlock

Class: ActionBlock	
This model class represents an action block. It loads/ saves data by communicating with the ActionChain object.	
Responsibility	**Collaborator**
Loads/ Exports the action block data.	ActionChain

Stores/ Updates the instruction type, arguments, and the internal counter.	

MazeData

Class: MazeData	
This model class represents a maze layout. Each cell is represented by a MazeItem. The maze and inventory are updated in real time by the GameplayScreen.	
Responsibility	**Collaborator**
Loads the maze data file.	MazeItem
Stores/ Updates/ Resets the active maze layout.	GameplayScreen
Stores/ Updates the inventory.	

MazeItem

Class: MazeItem	
This model class represents an item in a maze cell. It can be the spawn point, the exit, a key, a trap, or a wall.	
Responsibility	**Collaborator**
Stores/ Updates the type and coordinates of the maze item.	MazeData

HighScoreData

Class: HighScoreData	
This model class stores the top 5 high scores. It loads/ saves data using the high-score data file.	
Responsibility	**Collaborator**
Loads/ Updates the high-score data file.	HighScoreScreen (not drawn)